



NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

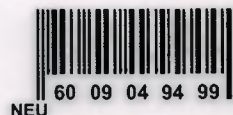
EDUCATIONAL ORGANIZATION SOFTWARE

**Graduation Project
COM400**

Student: Caner Çakır (20010391)

Supervisor: Mr. Ümit İlhan

Nicosia-2006





ACKNOWLEDGMENTS

For giving the chance of improving myself in this university with these conditions, I want to express my gratitude to Near East University.

First of all I want to thank my supervisor **Mr. Ümit İlhan** for his advices and his invaluable guidance in my Project . He forced my creativity to improve myself and my project. I am also grateful to him for giving his valuable time to me.

I also want to thank **Mr. Okan Donangil** who taught me the basics of programming approach and made me love the programming job.

Secondly I would like to thank all my friends especially **Muhammed Akgün and Aykut Danişman** who causes question marks in my mind and forced me to solve the problems I faced.

Especially I want to thank **Sebla Tanık** for being my inspiration and for her support.

At last My Family, the ones who deserves the most gratitude for giving their endless support and encouragement not only for my study, for all my life. I really thank you. It could not be done without you.

TABLE OF CONTENTS

ACKNOWLEDGEMENT	i
TABLE OF CONTENTS	ii
ABSTRACT	iv
INTRODUCTION	1
1. .NET FRAMEWORK	2
1.1 Introduction to .Net Framework	2
1.2 Description	3
1.3 How a Visual Basic Application is Compiled and Run?	3
2. VISUAL BASIC.NET	5
2.1 Why you should move to VB.NET?	5
2.2 Visual Basic.NET IDE	7
2.3 Project Files	16
2.4 Common Properties For Forms and Controls	16
3. OBJECT-ORIENTED PROGRAMMING	19
3.1 Introduction to Object-Oriented Programming	19
3.1.1 How to Refer to Properties , Methods and Events	20
3.1.2 How an Application Responds To Events	21
3.1.3 How to create an event procedure	22
4. MICROSOFT ACCESS	23
4.1 Introduction to Microsoft Access	23
4.2 Steps for Developing Database System	23
4.3 Using Microsoft Access	23
4.3.1 How to Open Access	23
4.3.2 How to Create a database	23
4.3.3 How to Open and Close a database	24
4.3.4 Creating a Table	24
4.3.5 Access Field Types	25
4.3.6 Primary Key	25
4.3.7 Building Relationship	26
4.3.8 Creating and Executing a Query	27
5. ADO.NET	28
5.1 Introduction to ADO.NET	28
5.2 How ADO.NET works?	28
5.3 Creating ADO.NET Objects	29
5.3.1 .NET data provider core objects	30
5.3.2 Connection, DataAdapter, Command	30
6. WEB SERVICES	40
6.1 Introduction to Web Services	40
6.2 How Web Services Work?	41
6.3 How to Add Web Services?	42
6.4 How to Write Web Service Methods?	46
7. ASP.NET	50
7.1 Introduction to Asp.NET	50

7.2 ASP.NET Advantages Over ASP	50
7.3 How ASP.NET Works?	51
7.4 Web Pages and Code	52
7.5 Basic Web Controls	53
7.6 First ASP.NET Application	54
7.7 Validation Controls	59
7.8 Types of Validators	63
7.9 The Databound ListControls Family	64
7.9.1 Using the Databound ListControls	65
8. EDUCATIONAL ORGANIZATION SOFTWARE	67
8.1 How Educational Organization Software Works?	67
8.2 Student Information System	82
CONCLUSION	90
REFERENCES	91
APPENDIX A: Program Codes (Only Service1.aspx)	92
APPENDIX B: Database Relationships	130

ABSTRACT

Automation programs with the development of the technology became compulsory software to make easy the works of the human in large platforms. Because the computers take place in every part of our lives, to equip them with programs that relieve our live is a good idea. Education Automation programs are one of these kind that help the personel, manager and owner in the Education sector to follow jobs easily and comfortably.

The main aim of this Project is making the users job easy. In this Project the student informations, course registrations and payments are recorded and pursued by the users. Also the teacher and advisor informations are kept in the storage. The registration is also applicable for the teachers to the courses. Project has a detailed search options for searching students, teachers and advisors seperately. These main application of the Project is designed and written in **Visual Basic .NET**.

Also this Project has a web based part that helps the students to reach the informations related. And give them to update and see list of informations about the courses like payment installment list and exam results. This project is prepared by using **ASP.NET**.

Also Microsoft Access DB are used to store all the information in different tables. The **Web Service** technology is used to hold the database in HTTP and to let the project to communicate with the database.

As a result this project combines the VB.NET and ASP.NET together with Web Service technology to allow user to work with the data on the HTTP.

INTRODUCTION

The Technology is changing very fast and the world is trying to adapt these changes. Lots of things are changed after internet was found. The interactive banking, communication over internet etc., these are the things that we can't imagine before 20 years.

Nowadays the technology in programming is developed and there are several programming languages and programming services and techniques. As internet becomes a need in daily life, also in programming internet became very important. Nowadays it becomes important to have strong Internet Programming abilities for any programmers. The industry is focusing on critical distributed computing with web services capabilities.

In my project web services are used to build a internet based educational windows application which works on the local machine. Since the database is in HTTP the software will work in any place where has an internet connection. It can be used both from one end of the world to the other end. Also a web page is build to let students have chance to analyze their informations from web.

CHAPTER ONE

1 .NET FRAMEWORK

1.1 Introduction to .Net Framework

.NET Framework provides a common set of services that application programs written in a .NET language such as Visual Basic .NET can use to run on various operating systems and hardware platforms. The .NET Framework is divided into two main components: the .NET Framework Class Library and the Common Language Runtime. The .NET Framework Class Library consists of segments of pre-written code called classes that provide many of the functions that you need for developing .NET applications. For instance, the Windows Forms classes are used for developing Windows Forms applications. The ASP.NET classes are used for developing Web Forms applications. And other classes let you work with databases, manage security, access files, and perform many other functions.

The Common Language Runtime, or CLR, provides the services that are needed for executing any application that's developed with one of the .NET languages. This is possible because all of the .NET languages compile to a common intermediate language.

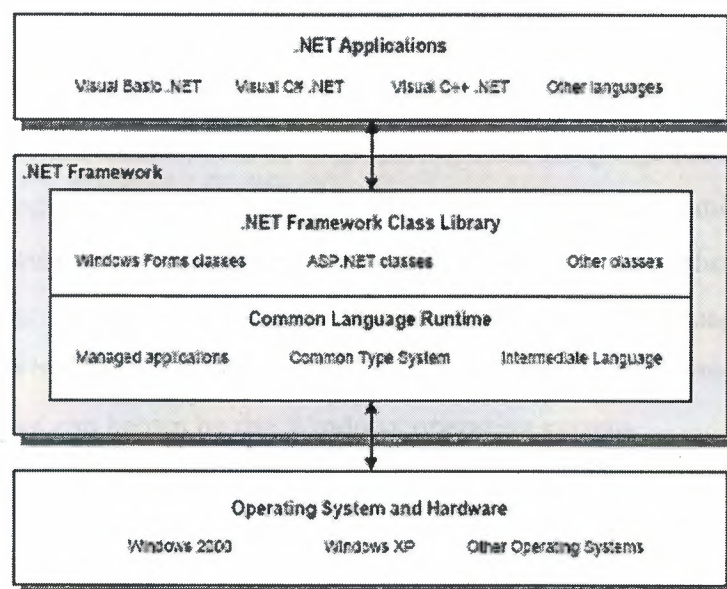


Figure 1.1.1 Main Components of .Net Framework

1.2 Description

- .NET applications do not access the operating system or computer hardware directly. Instead, they use services of the .NET Framework, which in turn access the operating system and hardware.
- The .NET Framework consists of two main components: the .NET Framework Class Library and the Common Language Runtime.
- The .NET Framework Class Library provides pre-written code in the form of classes that are available to all of the .NET programming languages. This class library consists of hundreds of classes, but you can create simple .NET applications once you learn how to use just a few of them.
- The Common Language Runtime, or CLR, is the foundation of the .NET Framework. It manages the execution of .NET programs by coordinating essential functions such as memory management, code execution, security, and other services. Because .NET applications are managed by the CLR, they are called managed applications.
- The Common Type System is a component of the CLR that ensures that all .NET applications use the same basic data types regardless of what programming languages were used to develop the applications.

1.3 How a Visual Basic Application is Compiled and Run?

Figure 1.3.1 shows how an application is compiled and run when using Visual Basic .NET. Visual Basic compiler is used, which is built into Visual Studio, to compile your Visual Basic source code into Microsoft Intermediate Language (or MSIL). For short, this can be referred to as Intermediate Language (or IL). The Intermediate Language is stored on disk in a file that's called an assembly. In addition to the IL, the assembly includes references to the classes that the application requires. The assembly can then be run on any PC that has the Common Language Runtime installed on it. When the assembly is run, the CLR converts the Intermediate Language to native code that can be run by the Windows operating system.

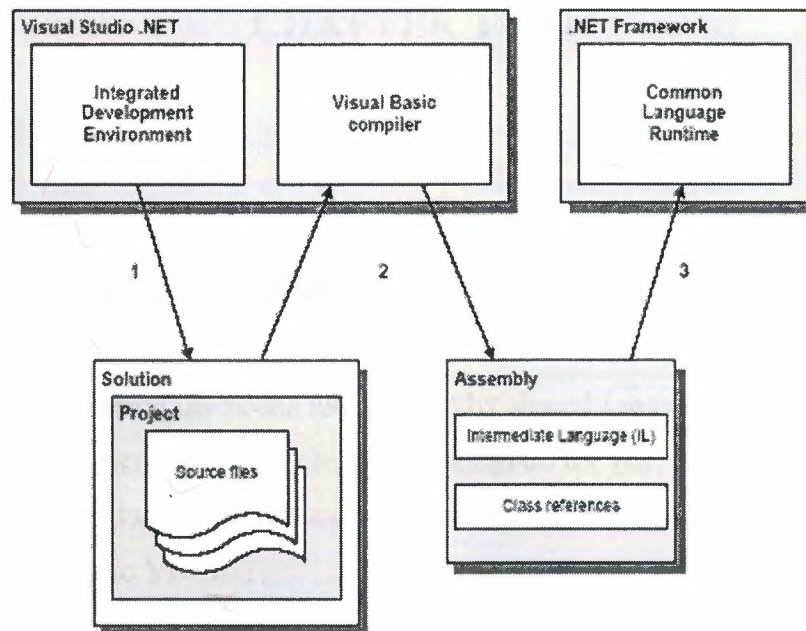


Figure 1.3.1 How a VB.Net Application is compiled and run

- The programmer uses Visual Studio's Integrated Development Environment to create a project, which includes one or more Visual Basic source files. In some cases, a project may contain other types of files, such as graphic image files or sound files. A solution is a container that holds projects. Although a solution can contain more than one project, the solution for most simple applications contains just one project. So you can think of the solution and the project as essentially the same thing.
- The Visual Basic compiler translates or builds the source code into Microsoft Intermediate Language (MSIL), or just Intermediate Language (IL). This language is stored on disk in an assembly that also contains references to the classes that the application requires. An assembly is simply an executable file that has an .exe or .dll extension.
- The assembly is then run by the .NET Framework's Common Language Runtime. The CLR manages all aspects of how the assembly is run, including converting the Intermediate Language to native code that can be run by the operating system, managing memory for the assembly, enforcing security, and so on.

CHAPTER TWO

2. VISUAL BASIC.NET

2.1 Why you should move to VB.NET?

One of the most common questions today is, "Why should I move to .NET?" .NET is new, and there are many questions about what it can do for you. From a Visual Basic standpoint, it's important to understand some of the dramatic benefits that can be achieved by moving to VB.NET.

Easy and GUI based programming language, making each & every task easier and improves programmer's productivity. VB has won the best RAD (Rapid Application Development) Tool award for three times and still keeping itself at top.

Microsoft first started Visual Basic in early 1990s and the project name was "Thunder". After the launch of VB 5.0, it crosses all the boundaries and won the best RAD Tool award by beating PowerBuilder in 1998. VB 5.0 came out with some great enhancements but definitely this time VB.NET has come with revolutionary changes to make it suitable for next generation of application development. Many developers today talk about new generation programming languages and don't count VB 6.0 as a powerful tool for developing good programs, but i would like to present the surprising data about VB developers given by Bill Gates.

"Since Visual Basic's inception, its community has grown to more than 3 million professional developers worldwide. In fact, about half the world's developers now use Visual Basic. The increasing power and richness of the PC provided the backbone for this amazing growth."

Certainly this seems to be an amazing thing to talk about these details in the world of Internet Programming and when we know that Java has already taken place of most popular Internet programming language, but this is hard truth. The only feature lacking in VB was its Internet capabilities and when we are moving towards the third generation of the Internet, it becomes important to have strong Internet Programming capabilities for any programming language.

The industry is focusing on critical distributed computing with web services capabilities. At this moment VB.NET is definitely a powerful tool to provide all these solutions in integrated environment of .NET technology. Let us discuss the major problems with VB 6.0, which has been creating troubles for VB developers for a long time.

Problems with VB 6.0

- No capabilities for multithreading.
- Lack of implementation inheritance and other object oriented features.
- Poor error handling capabilities.
- Poor integration with other languages such as C++.
- No effective user interface for Internet based applications.

In VB.NET all these shortcomings have been eliminated. In fact VB gets the most extensive changes of any existing language in the Visual Studio suite. Let us talk about the major features VB.NET has developed.

Some new features of VB.NET

- Full support for object oriented programming.
- Structured error handling capabilities.
- Access to .NET Framework.
- Powerful unified Integrated Development Environment (IDE).
- Inherent support for XML & Web Services.
- Better windows applications with Windows Forms.
- New Console capabilities of VB.NET.
- New Web capabilities with Web Forms.
- Immense power of tools & controls (including Server Controls).
- Interoperability with other .NET complined languages.
- Better database programming approach with ADO.NET.

... and many more. The list is very long. Now we have to figure out that how important role VB will play in future? The answer is there in the words of Bill Gates once again -

"The next 10 years will be an amazing time for software developers. The advancements in the way we develop, deploy, and use applications will be as profound as the architectural shift from DOS- to Windows-based programming. Visual Basic.NET will provide the foundation for building the solutions that enable a new age of truly distributed computing on the Internet.

Microsoft sees the Visual Basic community as a core part of this vision. If you're new to this community, I welcome you to what promises to be an incredibly exciting era. If you're a seasoned Visual Basic developer, I thank you for continuing to make it the world's most popular development tool. I'm confident that Visual Basic.NET will give you the power to write cutting-edge software for tomorrow's Internet."

2.2 Visual Basic.NET IDE

The new VB.NET IDE might look somewhat familiar to the Visual Basic developers, but there are some significant changes that make it a more useful environment.

However, these changes can be frustrating to experienced VB developers because many of the keystrokes have changed, windows have different names, and the debugging tools work differently. VB.NET is part of Visual Studio.NET (or VS.NET), which finally consolidates all the development languages into one place: VB.NET, C++.NET, C# and J#. You can even create a single solution, containing multiple projects, in which the individual projects are written in separate languages.

i. Start Page

The very first time you start Visual Studio.NET, you are taken to a screen that allows you to configure the IDE. That screen is the My Profile page. After your first visit to the My Profile page, all subsequent starts of Visual Studio.NET begin with the Start Page, as shown in **Figure 2.2.1**.

The start page contains a number of sections, as indicated by the links along the left side. These sections are:

- **Get Started** : This option allows opening a recent or existing project, or create a new one. No recent projects are listed on the Get Started area shown in **Figure 2.2.1** As you create projects in VB.NET, this area will display the four most recently opened projects. This area also contains links to open an existing project, to create a new project, and to log a bug report. Expect this last option to disappear after the final product is released.
- **What's New** : This option covers new language features in Visual Studio.NET, including each individual language and the Visual Studio.NET environment. There are links to topics in the help files on new features for the VS.NET languages, the .NET SDKs, and a link to check for VS.NET upgrades.
- **Online Community** : This provides links to the Microsoft newsgroups. These are newsgroups accessible with any newsreader, but they are served from Microsoft's news server (msnews.microsoft.com) and not normal Usenet news servers. This page appears blank in some of the interim builds of VS.NET, but expect it to be fixed for the released version of Beta 2.
- **Headlines** : Provides a place for links to news about .NET. In some interim builds of Beta 2, this page simply generates an error. However, by the time Beta 2 is released, expect this page to include a link to MSDN Online at least.
- **Search Online** : Searches the MSDN Online library.
- **My Profile** : This screen lets you choose the overall layout of Visual Studio.NET. You can set the keyboard mappings to the same scheme as in previous versions of Visual Studio, such as Visual Basic 6. You can also set the window layout to match previous versions of Visual Studio projects, and you can automatically filter help using the profile.

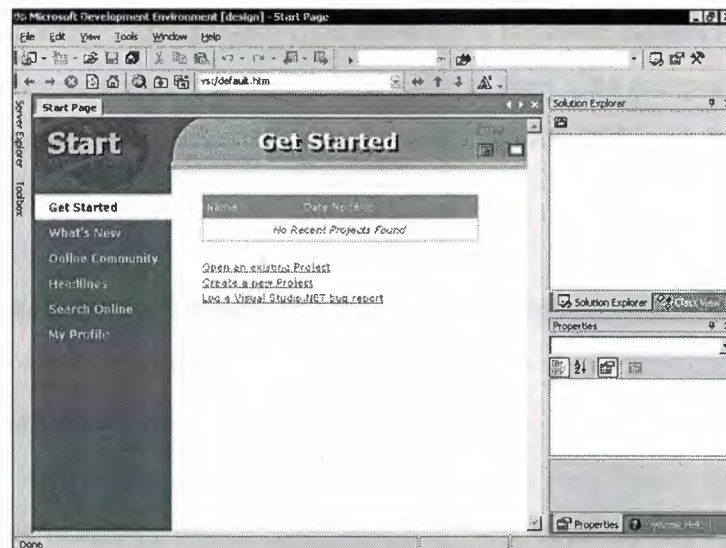


Figure 2.2.1: The Visual Studio.NET Start Page

ii. Creating a New Project

Return to the Start Page, identifiable by the tab at the top, and click on Get Started. Now, click on the Create New Project link. Doing so opens the New Project dialog shown in **Figure 2.2.2** Notice that there are different languages you can use to create applications in Visual Studio.NET.

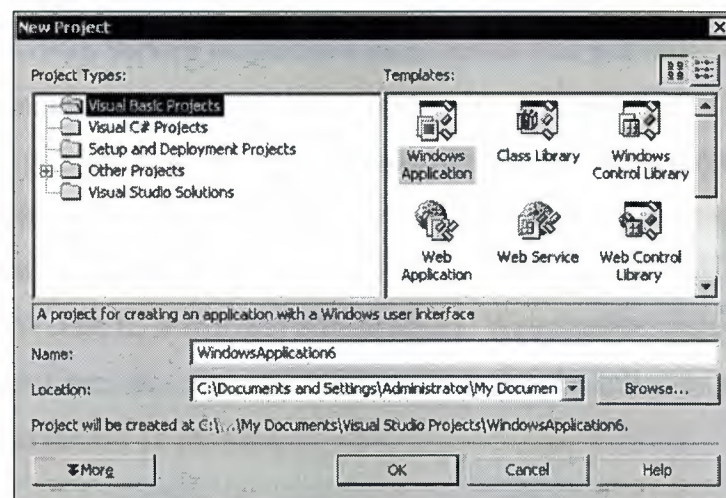


Figure 2.2.2 The New Project Dialog Box

If you examine the Visual Basic project types, you'll see that many of them are different from what you are used to with VB6. Some of the major project types are as follows:

- **Windows Application :** This is a standard executable, in VB6 terminology. It is the way to create applications with a Windows interface, using forms and controls. This is as close to "your father's VB" as you'll get in VB.NET.

- **Class Library :** This project type allows you to create classes that will be used in other applications. Think of it as similar to the COM components that you have been building, which VB6 called the ActiveX DLL and ActiveX EXE project types.
- **Windows Control Library :** This project type is for creating what used to be called ActiveX controls. This type allows you to create new controls to be used in Windows applications.
- **Web Application :** Visual Basic now has Web Application projects, which use ASP.NET to create dynamic Web applications. These projects allow you to create HTML, ASP.NET, and VB files. You will now code your Web applications using a powerful, event-driven model instead of the request/response model.
- **Web Service :** If you've used VB6 to create COM components and then made them available over HTTP with SOAP, you understand the concept of Web Services. Web Service projects are components that you make available to other applications via the Web; the underlying protocol is HTTP instead of DCOM, and you pass requests and receive responses behind the scenes using XML. Some of the major promises of Web Services are that they are all standards-based and are platform independent. Unlike DCOM, which was tied to a COM (that is, Windows) infrastructure, Web Service projects can be placed on any platform that supports .NET, and can then be called by any application using just the HTTP protocol.
- **Web Control Library :** As with Web Service projects, there's no exact match back in VB6 for the Web Control Library projects. Thanks to the new Web Application projects in VB.NET, you can add controls to Web pages just as you would in a standard Windows Application project, but VB.NET makes them HTML controls at runtime. You can design your own controls that can then be used by Web applications.

- **Console Application :** Many of the Windows administrative tools are still console (or command-line, or DOS) applications. Previously, you didn't have a good way to create console applications in VB, and instead had to rely on C++. Now, console applications are natively supported by VB.NET.
- **Windows Service :** As with console applications, there was no good way to create Windows services in previous versions of VB. Windows services, of course, are programs that run in the background of Windows, and can automatically start when the machine is booted, even if no one logs in. Those are the basic types of applications you can create. You can also create an Empty project (for Windows applications, class libraries, and services) or an empty Web Application (for Web applications).

iii. Examining the IDE

A new Windows Application is created and named "LearningVB". After a time, a new project will open up. Notice this adds a Form1.vb tab to the main window. In the main window, now there is an empty form. This is commonly referred to as the Form Designer. One difference that has occurred is that the files created have already been saved on your machine. In VB, you could create a project, do some quick coding, and then exit without saving, and nothing was stored on your machine. Now, however, the files are saved at creation, so each project you create does store something on the hard drive.

At the right side of the IDE, you'll see a window called the Solution Explorer. This works like the Project Explorer in VB6, showing you the projects and files you have in the current solution (what VB6 called a group). The Solution Explorer currently lists the solution name, the project name, and all the forms and modules. Right now, there is just one form, named Form1.vb. In addition, the window will have a file called AssemblyInfo.vb, which is part of the metadata that will be compiled into this assembly. You also see a new node, called References, in the list. If you expand the References node, you will see all the references that are already available to your project when you start. You can see the Solution Explorer in **Figure 2.2.3**.

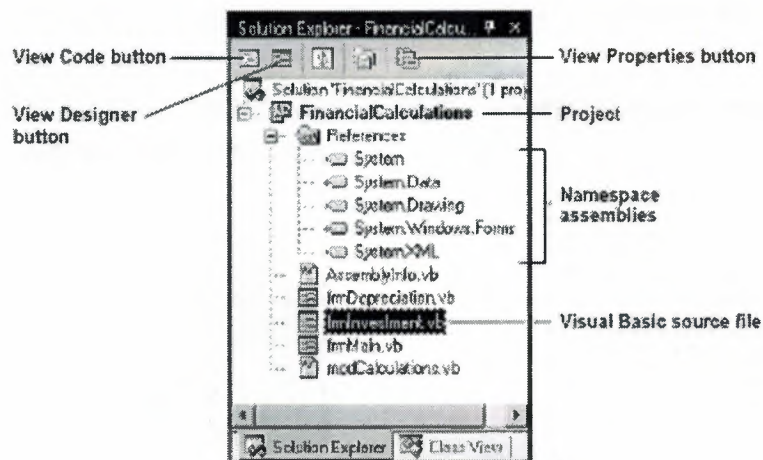


Figure 2.2.3 The Solution Explorer Window

Also Class View, exists at the bottom of the Server Explorer window. If you click on the Class View tab, you will see the LearningVB project listed. If you expand the project node, you will see the namespaces for this project listed. Expand the LearningVB namespace and you will see that just Form1 is listed below it. Expand Form1 and you will see some of the form's methods, as well as a node for Bases and Interfaces. If you expand that node and the Form node under it, you will see a long list of properties, methods, and events available to you in the form. (see a part of list in **Figure 2.2.4**)

If you want to know more about what one of those properties or methods can do for you, it's easy to look it up in the Object Browser. Properties of any method can be seen by scrolling down classes to find the method and by right clicking on it to choose Browse Definition. Object Browser opens as a tab in the main work area, and that you are on the definition for that method. Return Type and Father class is included in definition. (**Figure 2.2.5** shows the Class view)

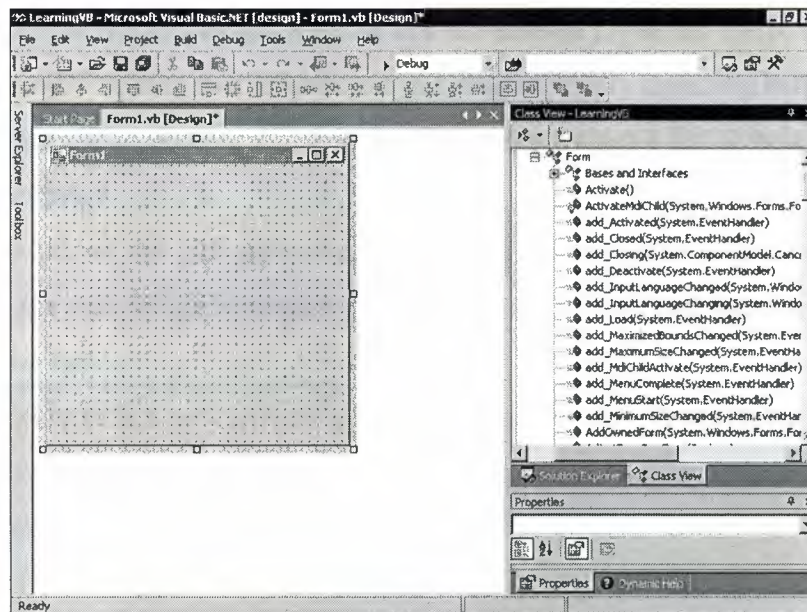


Figure 2.2.4 The New Class View Window

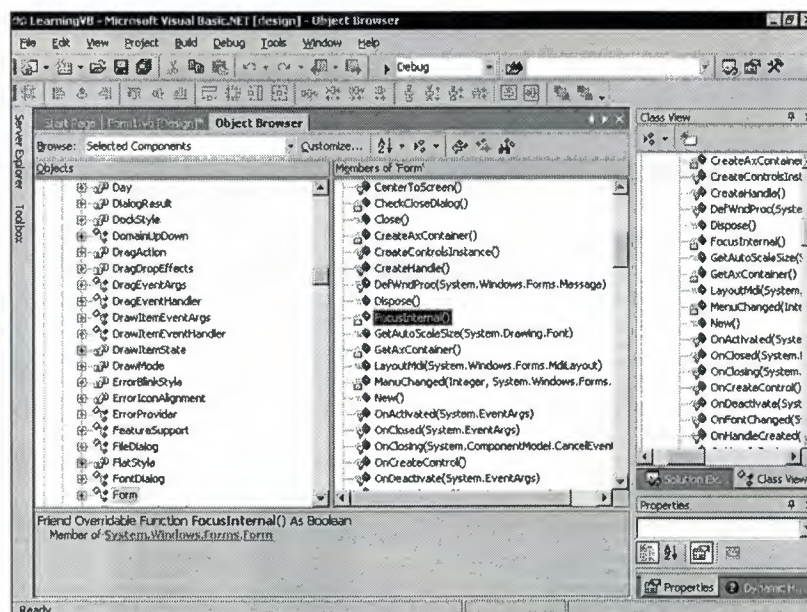


Figure 2.2.5 Object Browser in main work area

Below the Server Explorer/Class View windows is something that will be quite familiar to the VB Developers: the Properties window. If you close the Object Browser and go back to the Form1.vb [Design] tab, you should see the properties for Form1. You might actually have to click on the form for it to get the focus. After the form has the focus, you will see the properties for the form. Most of these properties will look very familiar to you, although there are some new ones.

In the same area as the Properties window is a tab labeled Dynamic Help. This is a new Visual Studio.NET feature that allows you to have constantly updating help while you work. It monitors what you are doing in the IDE and provides a list of help topics for your current activity.

Along the left side of the IDE are two sideways tabs. The first tab is labeled Server Explorer and the second tab is labeled Toolbox. To get either to appear, just hover the mouse over the tab.

The Server Explorer is a new feature to the IDE. It allows for discoverable services on various servers. For example, if you want to find machines that are running Microsoft SQL Server, there is a SQL Server Databases node under each server. (see **Figure 2.2.6**)

Toolbox is used to add controls to a form. The easiest way to do that is to click on the control in the Toolbox, then click the form at the location where you want to add the control. Then control can be resized by dragging one of the control's adjustment handles, and can be moved by dragging the it to a new location on the form.

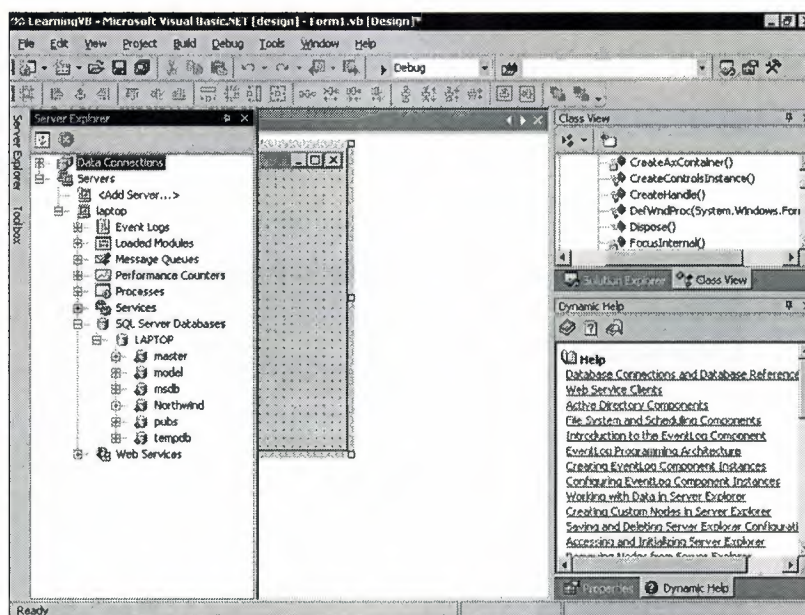


Figure 2.2.6 Server Explorer

The main part of the Visual Studio IDE contains one or more tabbed windows. To develop a form, you use the Form Designer window. And to develop code, you use the Code Editor window. A general view of form designer in Visual Basic .NET is shown in **Figure 2.2.7**.

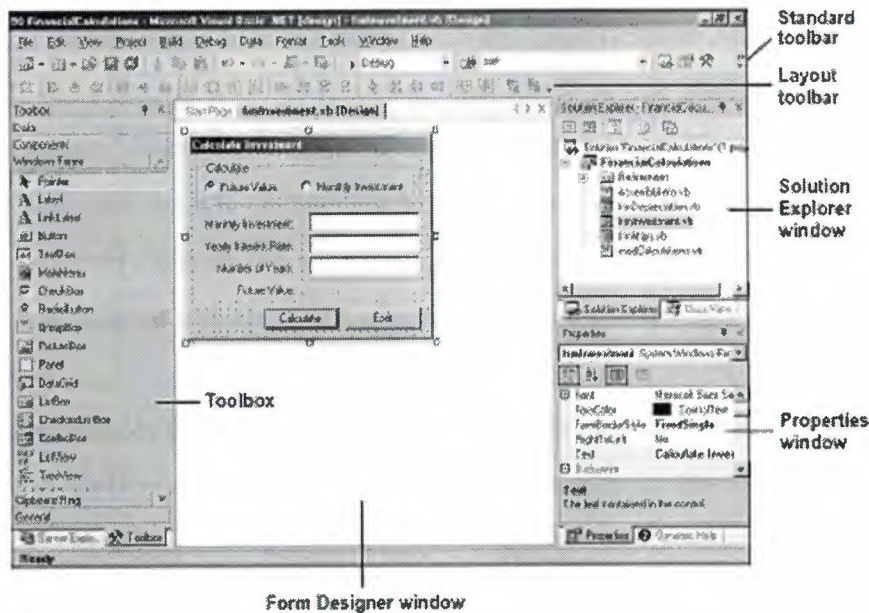


Figure 2.2.7 Form Designer and Windows

The Code Editor window is where you create and edit the Visual Basic code that your application requires. Code Editor can be displayed by double-clicking the form or one of the controls in the Form Designer window or by clicking the View Code button in the Solution Explorer. Moving beyond the views can be done by shortcuts Ctrl+Tab or Shift+Ctrl+Tab.

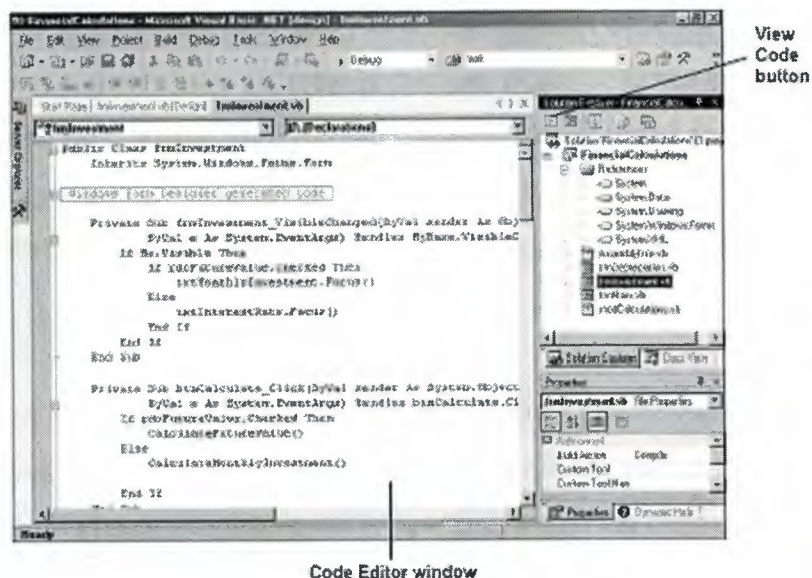


Figure 2.2.8 Code Editor Window

2.3 Project Files

- Visual Basic source files are stored with the file extension .vb. Each form you create for a project will have its own form file. You can also create code files that contain Visual Basic code but do not define a form. The Solution Explorer uses different icons to distinguish between form files and code files.
- The AssemblyInfo.vb file is created automatically when the project is created. It contains information about the assembly that's created when you compile the project.
- The References folder contains references to the assemblies for the namespaces that the application can use. These namespaces contain the classes that the project requires. In most cases, all the assemblies that you need are included when the project is created.
- In addition to the assemblies in the References folder, every Visual Basic application you develop has access to the Microsoft.VisualBasic assembly.

2.4 Common Properties For Forms and Controls

There are some common properties for forms and controls. The Name and the Text properties apply to both forms and controls. The other properties are presented in two groups: properties that apply to forms and properties that apply to controls. Note that some of the control properties only apply to certain types of controls. That's because different types of controls have different properties. Since all forms and controls must have a Name property, Visual Studio creates generic names for all forms and controls, such as Form1 or Button1. Often, though, you should change these generic names to something more meaningful, especially if you're going to refer to them in your Visual Basic code.

Forms and most controls also have a Text property that is visible when the form is displayed. A form's Text property is displayed in the form's title bar. For a control, the Text property is displayed somewhere within the control. The Text property of a button, for example, is displayed on the button, and the Text property of a text box is displayed in the text box.

i) The Name property

- Sets the name you use to identify a form or control in your Visual Basic code.
- Should only be changed if you intend to refer to the form or control in your code. For label controls whose values won't change during your program's execution, you can leave the name set to the default value.
- Use a specific prefix for naming the controls or forms so that the readability of the project will be easier.

ii) The Text property

- Sets the text that is displayed on the form or control. The default value is the form or control name, which you'll almost always want to change.
- For a form, the Text value is displayed in the title bar. For controls, the Text value is displayed directly on the control.
- For a text box, the Text value changes when the user types text into the field. As a result, you can use the Text property to access the information entered by the user.
- If you want a text box to be initially blank, be sure to clear its Text property

iii) Other Properties for Forms

Property	Description
AcceptButton	Identifies the button that will be activated when the user presses the Enter key.
CancelButton	Identifies the button that will be activated when the user presses the Esc key.
ControlBox	Determines whether a control box will be displayed in the upper left corner of the form.
FormBorderStyle	Sets the border style for the form.
MaximizeBox	Determines whether a Maximize button will be displayed on the form.
MinimizeBox	Determines whether a Minimize button will be displayed on the form.
StartPosition	Sets the position at which the form is displayed. To center the form, set this property to CenterScreen.

iv) Other Properties for Controls

Property	Description
BorderStyle	Sets the border style for controls.
Enabled	Determines whether the control will be enabled or disabled.
ReadOnly	Determines whether the text in some controls like text boxes can be edited.
TabIndex	Indicates the control's position in the tab order, which determines the order in which the controls will receive the focus when the user presses the Tab key.
TabStop	Determines whether the control will accept the focus when the user presses the Tab key to move from one control to another. Some controls, like labels, don't have the TabStop property because they can't receive the focus.
TextAlign	Sets the alignment for the text displayed on a control.

CHAPTER THREE

3. OBJECT-ORIENTED PROGRAMMING

3.1 Introduction to Object-Oriented Programming

Visual Basic .Net has Object-Oriented Programming structure. Each control on a form is an object, and the form itself is an object. These objects are derived from classes that are part of the .NET Class Library. By creating project actually a new class is created that inherits the characteristics of the Form class that's part of the .NET Class Library. And when the project is run actually an instance of your form class is created, which is known as an object. when a control is added to a form, actually a control object is added to the form. Each control is an instance of a specific class. For example, a text box control is an object that is an instance of the TextBox class.

The properties of an object define the object's characteristics and data. For instance, the Name property gives a name to a control. The methods of an object determine the operations that can be performed by the object. And an object's events are signals sent by the object to your application that something has happened that can be responded to.

i) Class and object concepts

- An object is a self-contained unit that combines code and data.
- A class is the code that defines the characteristics of an object. You can think of a class as a template for an object.
- An object is an instance of a class, and the process of creating an object from a class is called instantiation.
- More than one object instance can be created from a single class. For example, a form can have several button objects, all instantiated from the same Button class. Each is a separate object, but all share the characteristics of the Button class.
- A class can be based on an existing class. In that case, the existing class is referred to as the base class, and the new class inherits the characteristics of the base class.

ii) Property, method, and event concepts

- An object's interface consists of a clearly defined set of properties, methods, and events.
- The properties, methods, and events can be referred to as members of the object.
- Properties are the data associated with an object.
- Methods are the operations that an object can perform.
- Events are signals by which an object can notify other objects that something noteworthy has occurred.
- If you instantiate two or more instances of the same class, all of the objects have the same properties, methods, and events. However, the values assigned to the properties can vary from one instance to another.

3.1.1 How to Refer to Properties , Methods and Events

While entering the code for a form in the Code Editor window, often it is needed to refer to the properties, methods, and events of its objects. To do that, the name of the object is typed, a period (also known as a dot operator, or dot), and the name of the member. To make it easier to refer to the members of an object, Visual Basic provides the Auto List Members feature shown in this **Figure 3.1.1**. After an object name and a period is typed, this feature displays a list of the members that are available for that object.

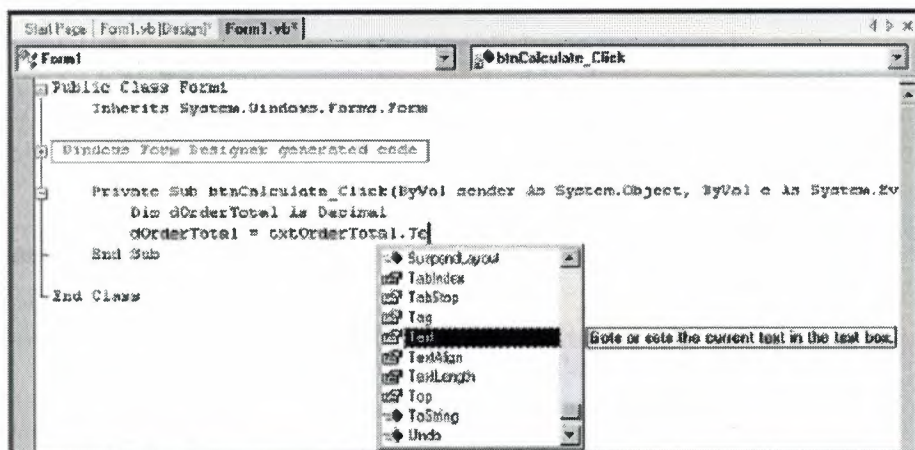


Figure 3.1.1 Auto List Members feature in Code Window

i) Statements that refer to properties

Txt1.Text = 10 Assigns the value 10 to the Text property of the textbox named Txt1.

Txt1.ReadOnly = True Assigns the True value to the ReadOnly property of the textbox named Txt1 so the user can't change its contents.

ii) Statements that refer to methods

txtMonthlyInvestment.Focus Uses the Focus method to move the focus to the text box named txtMonthlyInvestment.

Me.Close Uses the Close method to close the form that contains the method. In this example, Me is a keyword that is used to refer to the current instance of the class.

iii) Code that refers to an event

btnExit.Click Refers to the click event of a button named btnExit.

3.1.2 How an Application Responds To Events

Visual Basic applications are event-driven. That means they work by responding to the events that occur on objects. The event procedure may consists of several statements that's needed to perform desired job. The Private Sub and End Sub statements are generated by Visual Studio to mark the beginning and the end of the procedure.

i) Common control events

Event	Occurs when...
Click	...the user clicks on the control.
DblClick	...the user double-clicks on the control.
GotFocus	...the focus is moved to the control.
LostFocus	...the focus is moved from the control.

ii) Common form events

Event	Occurs when...
Load	...the form is loaded into memory.
Activated	...the form becomes the active form.
Closing	...the form is closing.
Closed	...the form is closed.

- Windows applications work by responding to events that occur on objects.
- To indicate how an application should respond to an event, you code an event procedure, which is also known as an event handler.
- An event can be an action that's initiated by the user like the Click event, or it can be an action initiated by program code like the Closed event.

3.1.3 How to create an event procedure

One way to create an event procedure is to select the object and event from the drop-down lists at the top of the window in Code Editor. Then Visual Studio generates the Sub and End Sub statements for you, and you can add the code for the procedure between those two statements.

You can also start an event procedure by double-clicking on an object in the Form Designer window. Then, Visual Studio opens the Code Editor window and generates Sub and End Sub statements for the default event of the object. (See **Figure 3.1.3**)

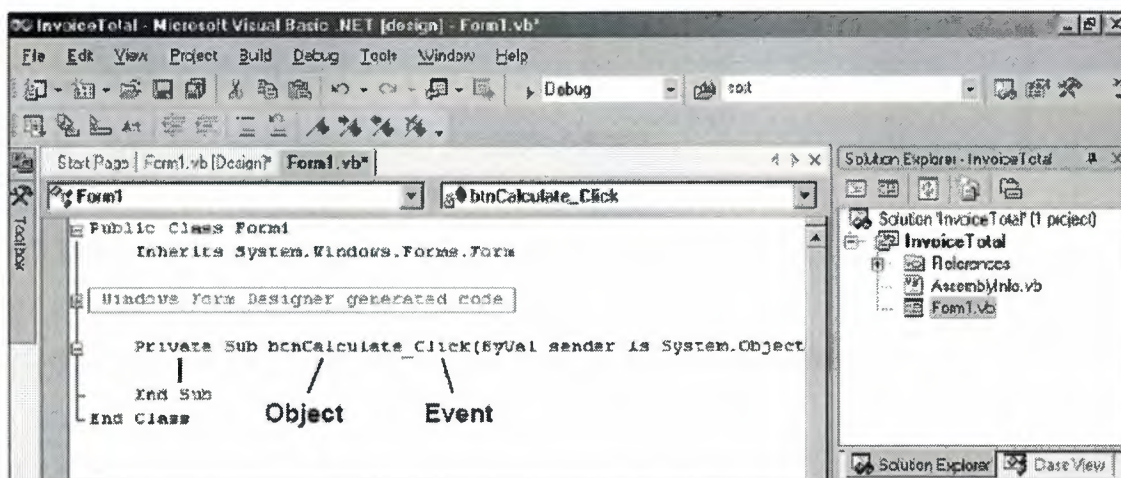


Figure 3.1.3 A Click event procedure

The Sub statement that's generated when you create an event procedure includes a Handles clause that names the object and event the procedure handles. The procedure name that's generated consists of the object name, an underscore, and the event name. Thus, btnCalculate_Click is the name of the procedure that handles the Click event of the btnCalculate button.

CHAPTER FOUR

4. MICROSOFT ACCESS

4.1 Introduction to Microsoft Access

Access is a **Relational Database Management System (RDMS)** that allows you to store, organize, and manipulate collections of information in an electronic format. A database is a collection of related information or data.

4.2 Steps for Developing Database System

These are the basic steps in designing a database:

- i. Determine the purpose of your database.
- ii. Determine the tables you need in the database.
- iii. Determine the fields you need in the tables.
- iv. Identify fields with unique values.
- v. Determine the relationships between tables.
- vi. Refine your design.
- vii. Add data and create other database objects.

4.3 Using Microsoft Access

This section will guide you to perform basic operations in Microsoft Access.

4.3.1 How to Open Access

- Click **Start** in Windows, select **All Programs** and click on **Microsoft Access**

4.3.2 How to Create a database

- Click **File, New** or click the new icon on the standard toolbar
- Select **Blank Database** from the Task Pane menu
- Type a name for database in the File Name window
- Click **Create**

4.3.3 How to Open and Close a database

To Open a database

- Click **File, Open** or click the open icon on the standard toolbar
- Browse to where the database is saved
- Click the name of the database
- Click **Open**

To Close a database

- Click **File, Close**

4.3.4 Creating a Table (Figure 4.3.4)

- Double-click **Create table in design view**
- In the Field Name column type the name of data field (i.e. FirstName)
- In the Data Type column select the type of data to be entered in the field (i.e. Text, Number, etc.)

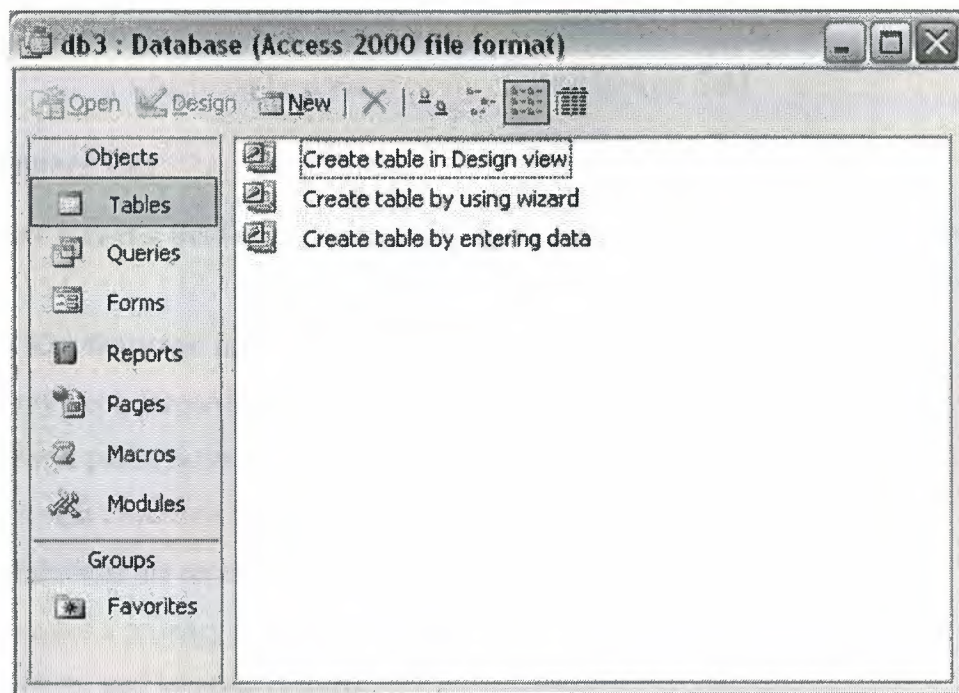


Figure 4.3.4 Database Tables Tab View

4.3.5 Access Field Types

Access Field Types	
Text	Up to 255 characters. Characters can be letters, numbers, and other marks, including punctuation.
Number	Number fields contain numbers (excluding currency amounts) that can be used in calculation.
Currency	Currency fields contain values up to four decimal places. Displays commas, dollar signs, and two digits to the right of the decimal point.
Date/Time	Date/Time fields utilize a variety of display formats. Can also be calculated.
Memo	Memo fields can contain up to 64,000 characters. The characters can be letters, numbers, and other marks, including punctuation marks.
AutoNumber	Used to automatically insert unique, sequential, or random numbers. Numbering typically begins with one.
Yes/No	Yes/No fields accept the following entries: Yes/No, True/False, or On/Off. Only one of the two values is displayed.
OLE/Object	OLE Object fields (object linking and embedding) contain objects created in other programs that can be linked or embedded.
Hyperlink	Hyperlink fields contain text or combinations of text and numbers. Used in forms, reports, and datasheets to jump to objects in the same or another database; to documents created with Word, Excel, and PowerPoint, and URLs.
Lookup Wizard	Lookup fields enable the user to choose a value from another table or from a list of values. Choosing this option starts the Lookup Wizard, which can be used to easily create a lookup field.

4.3.6 Primary Key

A **Primary Key** is a field or combination of fields that uniquely identify each record in a table.


Primary Key features: no two records in a table can have the same value in the primary key field. Records are automatically sorted based on the primary key.

Primary Keys perform the following functions:

- Prevent duplicate values.
- Maintains the record order.
- Creates a primary index: indexes are used to improve the speed of queries, reports, and locating records.
- Facilitates relationships to other normalized data in the database. The primary table to be joined must have a primary key field.

Note: Memo, Yes/No, OLE object, and Hyperlink fields can't be Primary fields.

4.3.7 Building Relationship

To build a relationship between tables the button “Relationships”  is pressed in the toolbar.

Relation is built in below 5 steps:

- i. Right click on Relationships Window and click the “Show Table...” Button
- ii. Select tables and click the Add button to add tables on Relationships Window
- iii. To build a relation between any tables, the primary key of one table should be foreign key of another table. To do that just drag primary key field to the proper field of other tables and drop.
- iv. Edit Relationships window will appear and click “Enforce Referential Integrity”.
- v. Click close.

Then the Relationships Window will look like in **Figure 4.3.7**.

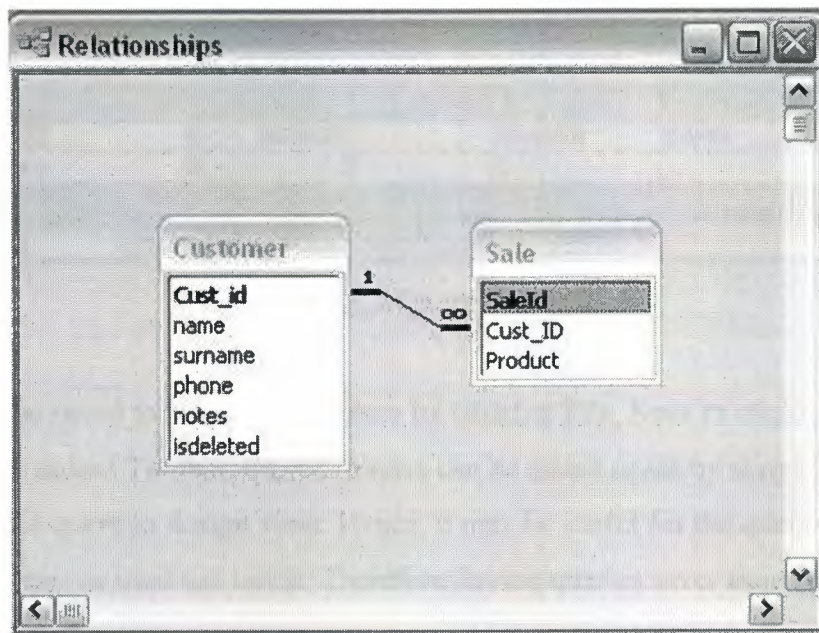



Figure 4.3.7 Relationships Window

4.3.8 Creating and Executing a Query

- Double-click “Create query in design view”
- Select and add tables in the “Show Table” window
- Right click on the query and write your sql statement. (see **Figure 4.3.8.1**)
- Execute the sql statement by clicking the “Run” button  on the standard toolbar.

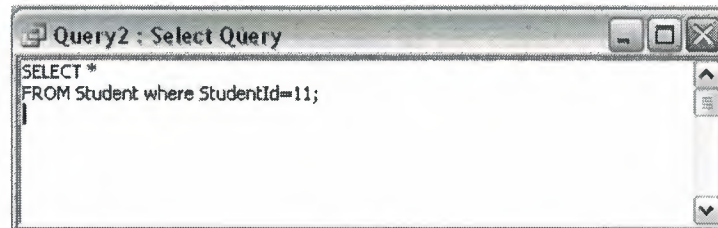
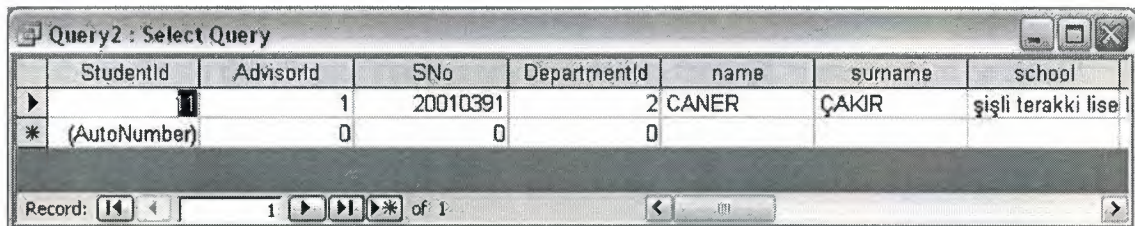


Figure 4.3.8.1 Query View

Above query’s result is as shown in **Figure 4.3.8.2**.



StudentId	AdvisorId	SNo	DepartmentId	name	surname	school
11	1	20010391	2	CANER	ÇAKIR	şişli terakki lise
*(AutoNumber)	0	0	0			

Figure 4.3.8.2 Query Result View

Queries can be saved to recall in the future by clicking **File, Save** or clicking the save icon on the Standard Toolbar. Stored queries can be called again by simply double clicking on the query in design view. Hence, it may be useful for the queries which are run several times or used too much. Therefore Saving queries saves time from writing queries again and again.

CHAPTER FIVE

5. ADO.NET

5.1 Introduction to ADO.NET

ADO.NET (ActiveX Data Objects .NET) is the primary data access API for the .NET Framework. It provides the classes that you use as you develop database applications with Visual Basic .NET as well as other .NET languages.

5.2 How ADO.NET works?

To work with data using ADO.NET, a variety of ADO.NET objects are used (see **Figure 5.2**). To start, the data used by an application is stored in a **dataset** that contains one or more **data tables**. To load data into a data table, you use a **data adapter**. The main function of the data adapter is to manage the flow of data between a dataset and a database. To do that, it uses **commands** that define the SQL statements to be issued. The command for retrieving data, for example, typically defines a Select statement. Then, the command connects to the database using a **connection** and passes the Select statement to the database. After the Select statement is executed, the result set it produces is sent back to the data adapter, which stores the results in the data table.

To update the data in a database, the data adapter uses a command that defines an Insert, Update, or Delete statement for a data table. Then, the command connects to the database and performs the requested operation.

The data in a dataset is independent of the database that the data was retrieved from. In fact, the connection to the database is typically closed after the data is retrieved from the database. Then, the connection is opened again when it's needed. Because of that, the application must work with the copy of the data that's stored in the dataset. The architecture that's used to implement this type of data processing is referred to as a **disconnected data architecture**.

One of the advantages of using a disconnected data architecture is improved system performance due to the use of fewer system resources for maintaining connections. Another advantage is that it makes ADO.NET compatible with ASP.NET web applications.

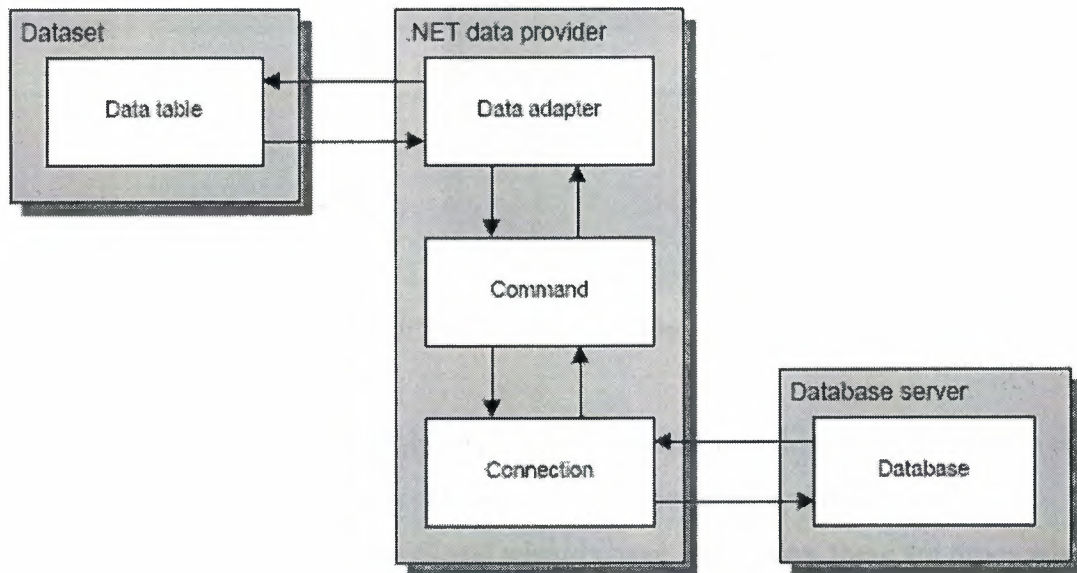


Figure 5.2 Basic ADO.NET objects

5.3 Creating ADO.NET Objects

Two basic techniques are used to create the ADO.NET objects while developing database applications. First, method is to use the components in the Data tab of the Toolbox to create ADO.NET objects by dragging and dropping them onto a form. Notice that the names of most of the components in the Data tab are prefixed with either “OleDb” or “Sql”. (see **Figure 5.3.1**)

The second technique for creating ADO.NET objects is to write the code yourself. The code shown in **figure 5.3.b**, for example, creates three objects: a connection named conPayables, a data adapter named daVendors, and a dataset named dsPayables. It also uses the Fill method of the data adapter to retrieve data from the database identified by the connection and load it into the dataset.



Figure 5.3.1 ADO.NET Objects in Data Tab of the Toolbox

Although creating ADO.NET objects through code is more time-consuming than using the components and wizards, it can result in more compact and efficient code. In addition, because the components and wizards have limitations, there are times when it is needed to write code.

5.3.1 .NET data provider core objects

Object	Description
Connection	Establishes a connection to a database.
Command	Represents an individual SQL statement that can be executed against the database.
Data reader	Provides read-only, forward-only access to the data in a database.
Data adapter	Provides the link between the command and connection objects and a dataset object.

5.3.2 Connection, DataAdapter, Command

As it is discussed in previous sections , ADO.NET objects can be created by dragging and dropping them onto a form.

i. Connection Creation in Design View

- OleDbConnection is found in Data Tab and dropped onto the form. As it is appeared on the form the properties of the connection are listed in the Properties Window. (See **Figure 5.3.2.1**)
- At first sight on the Properties Window, **Name** and **ConnectionString** properties appear.

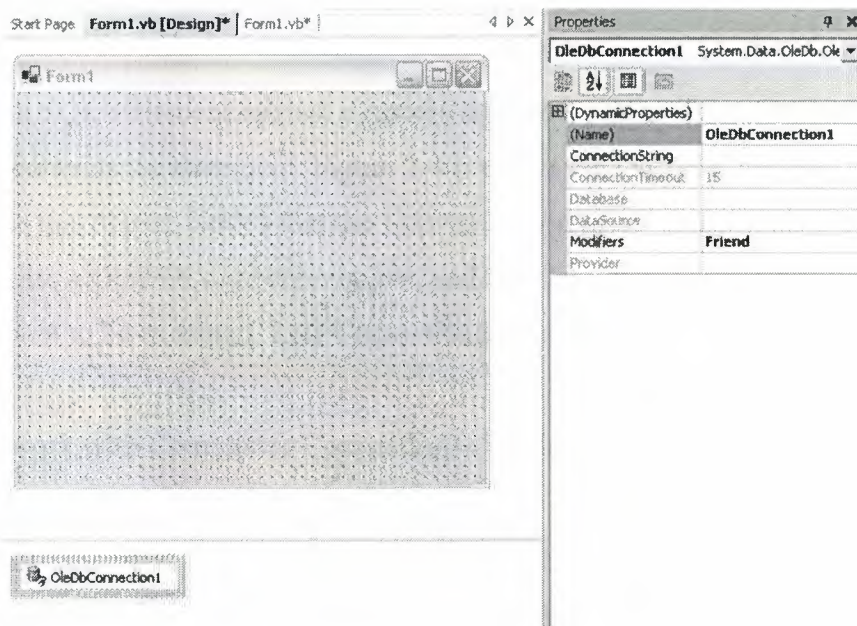


Figure 5.3.2.1 Connection and Properties Window

- Name Property can be changed to the desired unique name.
- To set ConnectionString property click on the combobox besides the ConnectionString field and click **<New Connection...>**.
- VB Developers will be familiar with the wizard that appears, Click **Provider** Tab and set the provider as "Microsoft Jet 4.0 OLE DB Provider".
- By Clicking **Connection** Tab, setting the database Url and clicking the **"Include password"** button on the opened dialogbox, The Connection settings are finished.

Common properties and methods of the Connection class

Property	Description
ConnectionString	Contains information that lets you connect to an Access database. The connection string includes information such as the name of the server, the name of the database, and login information.
Method	Description
Open	Opens a connection to a database.
Close	Closes a connection to a database.

ii. Command Creation in Design View

To execute a SQL statement against a Access database, you create a OleDbCommand object that contains the statement. Figure 2-4 presents the OleDbCommand class you use to create this object. Notice that the Connection property of this class associates the command with a OleDbConnection object, and the CommandText property contains the SQL statement to be executed. Lets Create **OleDbCommand**:

- To work with Commands Connection is needed so as it is described in previous sections Connection is created.
- OleDbCommand is found in Data Tab and dropped onto the form. As it is appeared on the form the properties of the connection are listed in the Properties Window. (See **Figure 5.3.2.2**)
- At first sight on the Properties Window, **Name**, **CommandText** and **Connection** properties appear.
- Name Property can be changed to the desired unique name.
- Connection is set by selecting a existing connection (which is created in the first step) or a new connection.
- Sql statement will be either written by just writing statements in the CommandText Property field of the Command or by clicking the “...” button besides the CommandText field so that a QueryBuilder will appear and Sql statement is written on it as described in previous Chapter .

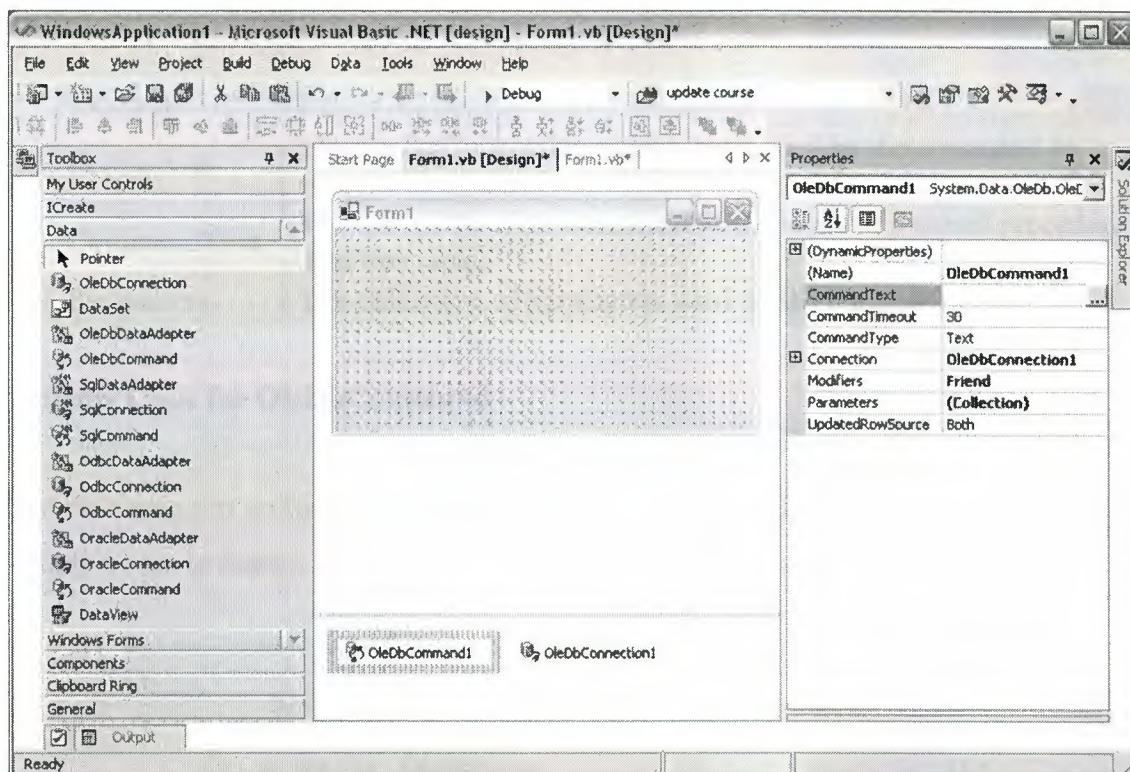


Figure 5.3.2.2 OleDbCommand and Properties Window

Executing a command object is done directly by using one of the three Execute methods shown in **Figure 5.3.2.3**. If the command contains a Select statement, for example, you can execute it using either **ExecuteReader** or **ExecuteScalar**. If you use **ExecuteReader**, the results are returned as a **DataReader** object. If you use **ExecuteScalar**, only the value in the first column and row of the query results is returned.

If the command contains an Insert, Update, or Delete statement, you'll use **ExecuteNonQuery** method to execute it. This method returns an integer value that indicates the number of rows that were affected by the command. For example, the command deletes a single row, the **ExecuteNonQuery** method returns 1.

Method	Description
ExecuteReader	Executes a query and returns the result as a OleDbDataReader object.
ExecuteNonQuery	Executes the command and returns an integer representing the number of rows affected.
ExecuteScalar	Executes a query and returns the first column of the first row returned by the query.

Figure 5.3.2.3 Methods of the OleDbCommand Class

Common properties and methods of the Command class

Property	Description
Connection	The OleDbConnection object that's used by the command to connect to the database.
CommandText	The text of the SQL command or the name of a stored procedure or database table.
Parameters	The collection of parameters used by the command.

Writing Code for OleDbCommand

Before coming to writing code , the Connection, CommandText, Parameters properties should be set in design view. Then an Command can be executed as shown below

```
'Below Codes are fixed for the command created in design view
Try
    OleDbConneciton1.Open()           'Open Connection
    OleDbCommand1.ExecuteNonQuery()   'Execute Command
Catch ex As Exception                 'Catch Errors
    MsgBox(ex.Message)                 'Show Errors if exists
Finally
    OleDbConnection1.Close()          'Close Connection
End Try
```

OleDbCommand can be created and executed by code only as shown below:

```
'Declare Command
Dim mycommand As New OleDb.OleDbCommand
'Set Command's Connection
mycommand.Connection = oledbconnection1
'Set CommandText of Command which will be performed
mycommand.CommandText = "Update City set CityName=@name"
'Add Parameter to the command that is required
mycommand.Parameters.Add("@name",name.text)
Try
    OleDbConnection1.Open()           'Open Connection
    mycommand.ExecuteNonQuery()       'Execute Command
Catch ex As Exception                 'Catch Errors
    MsgBox(ex.Message)                 'Show Errors if exists
Finally
    OleDbConnection1.Close()          'Close Connection
End Try
```

iii. Data Adapter Creation in Design View

The job of a data adapter is to provide a link between a database and a dataset. The four properties of the **OleDbDataAdapter** class listed in **Figure 5.3.2.4** identify the four SQL commands that the data adapter uses to transfer data from the database to the dataset and vice versa. The **SelectCommand** property identifies the command object that's used to retrieve data from the database. And the **DeleteCommand**, **InsertCommand**, and **UpdateCommand** properties identify the commands that are used to update the database based on changes made to the data in the dataset.

To execute the command identified by the **SelectCommand** property and place the data that's retrieved in a dataset, you use the **Fill** method.

(Methods of the **OleDbDataAdapter** class listed in **Figure 5.3.2.5**) Then, the application can work with the data in the dataset without affecting the data in the database.

Property	Description
SelectCommand	A SqlCommand object representing the Select statement used to query the database.
DeleteCommand	A SqlCommand object representing the Delete statement used to delete a row from the database.
InsertCommand	A SqlCommand object representing the Insert statement used to add a row to the database.
UpdateCommand	A SqlCommand object representing the Update statement used to update a row in the database.

Figure 5.3.2.4 The four properties of the OleDbDataAdapter

Method	Description
Fill	Executes the command identified by the SelectCommand property and loads the result into a dataset object.
Update	Executes the commands identified by the DeleteCommand, InsertCommand, and UpdateCommand properties for each row in the dataset that was deleted, added, or updated.

Figure 5.3.2.5 The methods of the OleDbDataAdapter

Creating OleDbDataAdapter is very simple and straight forward:

- As OleDbDataAdapter is dragged and dropped onto form, a wizard called DataAdapter Configuration Wizard appears. Click "Next".
- Choose Which Data Connection the data adapter will use. Click "Next". (see **Figure 5.3.2.6**)

- Choose “Use Sql Statements”. After that choice a blank field will be ready to hold sql statements. You Either write your Sql Statements in that field or click Query Builder to not to deal with Sql if you are not good at it. Click “Next”.
- Last Window will come from wizard that shows you if the Sql statements are correct or if there is an error or warning about statements. Click “Finish”.

As OleDbDataAdapter is created on the form, you will notice that automatically an OleDbConnection is created on the form. (Connection contains the database selected in the wizard.)

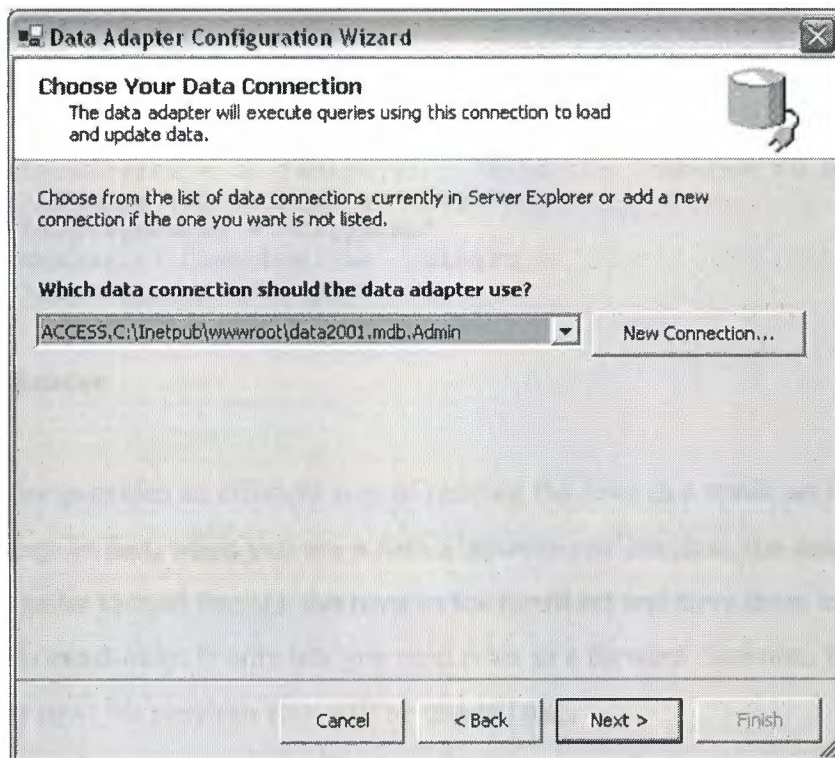


Figure 5.3.2.6 Data Adapter Configuration Wizard

Writing Code for OleDbDataAdapter

Below Code will list all cities into the combobox. DisplayMember means what user will see on combobox and ValueMember holds the CityId that when user select a city at back the CityId is selected. That is useful for Delete,Update operations.


```

Dim ds As New DataSet                                'Create a new Dataset
'Fill Dataset with result of the DataAdapter
OleDbDataAdapter1.Fill(ds)
ComboBox1.DataSource = ds.Tables(0)                  'Bind the Combobox to
dataset
'Set DisplayMember ofcombobox1 as City Name
ComboBox1.DisplayMember = "CityName"
'Set ValueMember of Combobox1 as CityId
ComboBox1.ValueMember = "CityId"

```

OleDbDataAdapter can be created and executed in the code as shown below:

```

'Create DataAdapter with Sql Statement and Connection
Dim MyAdapter As New OleDb.OleDbDataAdapter("Select * from City",
oledbconnection1)
Dim ds As New DataSet                                'Create a new Dataset
'Fill Dataset with result of the DataAdapter
MyAdapter.Fill(ds)

ComboBox1.DataSource = ds.Tables(0)                  'Bind the Combobox to dataset
'Set DisplayMember ofcombobox1 as City Name
ComboBox1.DisplayMember = "CityName"
'Set ValueMember of Combobox1 as CityId
ComboBox1.ValueMember = "CityId"

```

iv. Data Reader

A **data reader** provides an efficient way of reading the rows in a result set returned by a database query. In fact, when you use a data adapter to retrieve data, the data adapter uses a data reader to read through the rows in the result set and store them in a dataset. Data reader is **read-only**. It only lets you read rows in a forward direction. Once you read the next row, the previous row will be unavailable.

In most cases, it is used to code the Read method in a loop that reads and processes rows until the end of the data reader is reached. To access a column of data from the current row of a data reader, the Item property is used. Common properties and methods of the DataReader class is shown in **Figure 5.3.2.7**

To identify the column, either its index value is used as below:

```
drCustomer.Item(0)
```

or its name is used as below:

```
drCustomer.Item("Name")
```

Property	Description
Item	Accesses the column with the specified index or name from the current row.
FieldCount	The number of columns in the current row.
Method	Description
Read	Reads the next row. Returns True if there are more rows. Otherwise, returns False.
Close	Closes the data reader.

Figure 5.3.2.7 Common properties and methods of the DataReader class

v. Dataset

Dataset is structured much like a relational database. It can contain one or more tables, and each table can contain one or more columns and rows. In addition, each table can contain one or more constraints that can define a unique key within the table or a foreign key of another table in the dataset. If a dataset contains two or more tables, the dataset can also define the relationships between those tables. (see **Figure 5.3.2.8**)

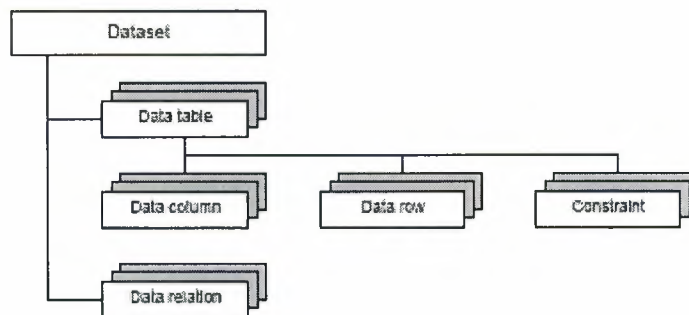


Figure 5.3.2.8 The basic organization of an ADO.NET dataset.

Each Table collection has a Count property that you can use to determine how many items are in the collection. To get the number of tables in a dataset named ds, for example, below code is used:

```
ds.Tables.Count()
```

To access a specific item in a collection, Item property is used. On that property, it is required to specify the index value or name of the item which is desired to access.

```
ds.Tables.Item("Customer") or ds.Tables.Item(0)
```

Since Item is the default property of the collection class, however, you typically omit it like this:

`ds.Tables("Customer")` or `ds.Tables(0)`

The Common properties and methods of the DataSet class is shown in **Figure 5.3.2.9**

Property	Description
DataSetName	The name of the dataset.
Tables	A collection of the DataTable objects contained in the dataset.
Relations	A collection of the DataRelation objects contained in the dataset.

Figure 5.3.2.9 Properties and methods of the DataSet

CHAPTER SIX

6. WEB SERVICES

6.1 Introduction to Web Services

Microsoft likes to point out that .NET acts like a huge operating system. In effect, the entire Internet becomes your operating system. This means that pieces of your applications can be **distributed over the Internet** but the applications run as if the pieces were all on your local machine.

Imagine if you had told someone back in the early days of Visual Basic that someday they'd be writing their applications in a number of separate components and putting those parts on different machines. The application sitting on the user's desktop would call these components on other machines, and those components would access the data on still other machines. The data would be returned to these components and finally flow back to the client application.

Naturally, this sounds quite normal today. However, now consider taking those components, and even the database, and removing them from your internal network. Spread them out all over the Internet, so that the only way with which you can communicate with them is **HTTP**. This is precisely what a Web service is all about. The idea behind a Web service is to create a reusable component that can be called over standard HTTP, but has the full power of a .NET language application. These components are discoverable, which means that you can locate and call available components. The format for calling particular methods is exposed as well, so anyone can determine what methods are available and how to call them.

6.2 How Web Services Work?

Every Method in Web Services is made public by adding the **<WebMethod(>** attribute to the method declaration. This makes the method automatically discoverable by anyone accessing the project's URL. Any class that has one or more methods marked with **<WebMethod(>** becomes a Web service. The Framework handles the task of setting up all the necessary hooks for the component to be callable via HTTP.

The full `System.Web.Services.myService` syntax is used to call predefined services.

By Adding a Web Service project below files are created in the project:

- `Global.asax`
- `Web.config`
- `<Service_Name>.asmx`
- `AssemblyInfo.vb`

The **Global.asax** file, is an optional file that contains code for responding to application-level events raised by ASP.NET or by `HttpModules`. The `Global.asax` file itself is configured so that any direct URL request for it is automatically rejected; external users cannot download or view the code written within it.

Web.config files are Web Forms configuration files, provide settings for every Web Forms page in the same directory as the configuration file. The settings are usually also inherited by subdirectories.

<Service_Name>.asmx file contains the **WebService** processing directive and serves as the addressable entry point for the XML Web service. The `<Service_Name>.asmx.vb` class file is a hidden, dependent file of `<Service_Name>.asmx`. It contains the code-behind class for the XML Web service. When viewing the Code View of `<Service_Name>.asmx`, you see the contents of this file.

A project information file **AssemblyInfo.vb** that contains metadata about the assemblies in a project, such as name, version, and culture information.

6.3 How to Add Web Services?

- Right Click on the solution name on Solution Explorer and click Add -->New Project or simply click on the File --> New Project.
- Select ASP.NET Web Service , you will see Location field below. Location for webservices are always in the localhost without changing the localhost change the name of the web service. See **Figure 6.3.1**

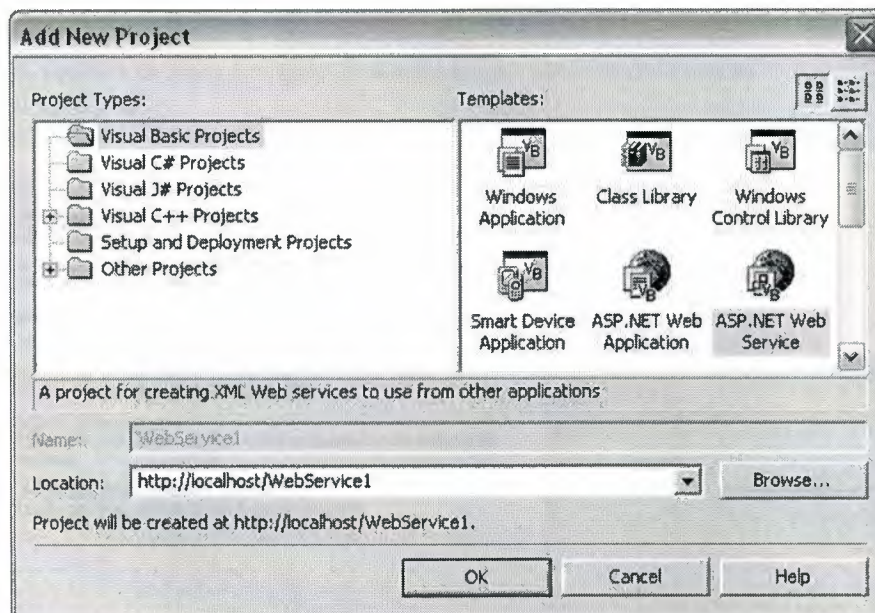


Figure 6.3.1 Add New ASP.NET Web Service

After adding **Web Service** , VB.NET will create a **virtual directory** on the localhost (**c:\inetpub\wwwroot\...**). And for every open process of this project VB.NET will communicate with these virtual directories to import and update web service.

Actually this is not enough to work with the Web Services. In order to reach and communicate with the web services from any project (Windows applications or Web applications) it is needed to add **Web Reference** to the current project which will communicate with Web Service.

To add Web Reference below steps should be followed:

- Right click on the “**References**” in the project which you want to communicate with the web services. Click “**Add Web Reference...**”
- A window appear and waits for a url for the web service. You can either write Url to the proper field or you can use the “**Browse to**” Links to find where the web service is. (see **Figure 6.3.2**)

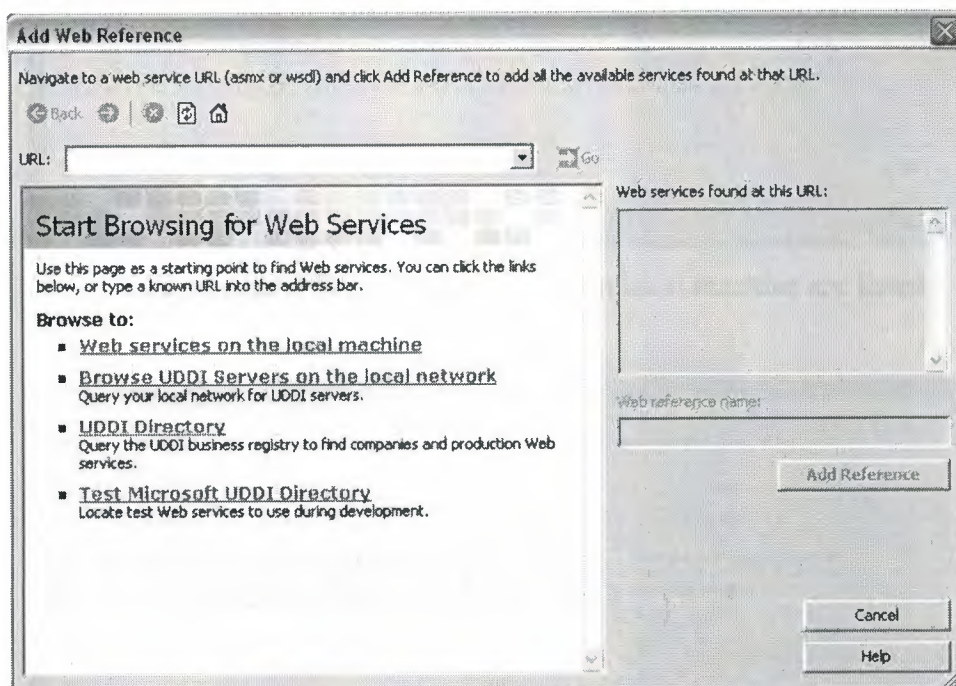
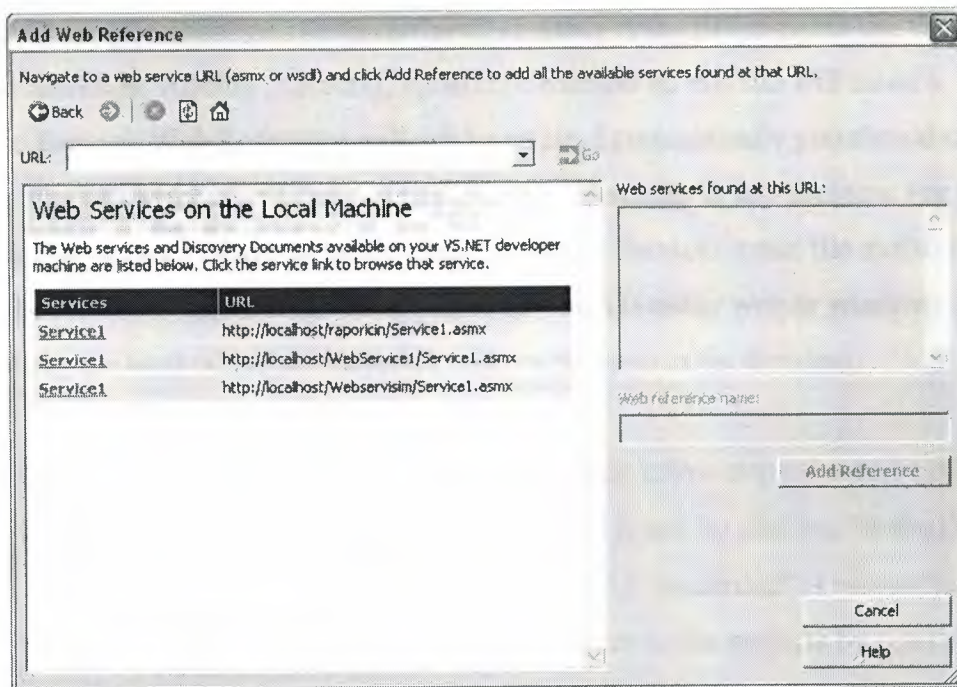


Figure 6.3.2 Browse Web Service

- By Clicking on the “**Web services on the local machine**” the web services are searched on the local machine and found services are listed. (see **Figure 6.3.3**)
- Selecting a web service on the list will bring all methods listed on the window and the service description. Methods can be tested by clicking on them and write the parameters to the fields required for the method. (see **Figure 6.3.4**)
- Web reference name is default “**localhost**”, it can be changed by writing a unique name in the “**Web reference name**”.
- After changing the name of the web reference you can add it by clicking on the “**Add Reference**” button.



See Figure 6.3.3 Web Services found on local machine are listed

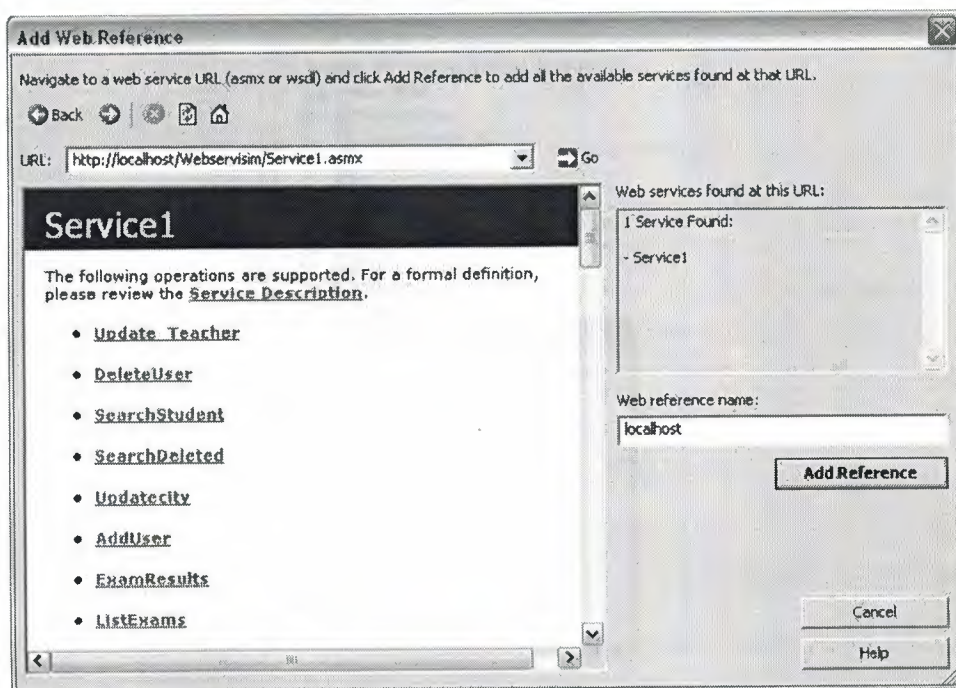


Figure 6.3.4 Web Service methods listed

After adding the web reference to the project there will be a new tree node named “Web References” and the web references will be grouped together there.

As it is discussed in previous sections **Service1.asmx** is the file contains the **Web Service methods**. Adding , deleting, updating a method on this file will cause a problem. Because Web Reference will not be updated automatically you should save and rebuild the web service first then update the web references in any projects. For example if a method named method1 is written in the Service1.asmx file and saved. When the service is called from the application which is either web or windows application, the method named method1 will not be seen in the Service1.

Hence After any change done to the Service1.asmx file below steps should be followed:

- **Rebuild** the Web Service by right clicking on it and by clicking “**Rebuild**”.
- After the message on the status bar “Rebuild All succeeded” is recognized then **Update** the Web References which are included to the projects by right clicking on the web reference and clicking “**Update**”. (see **Figure 6.3.5**)

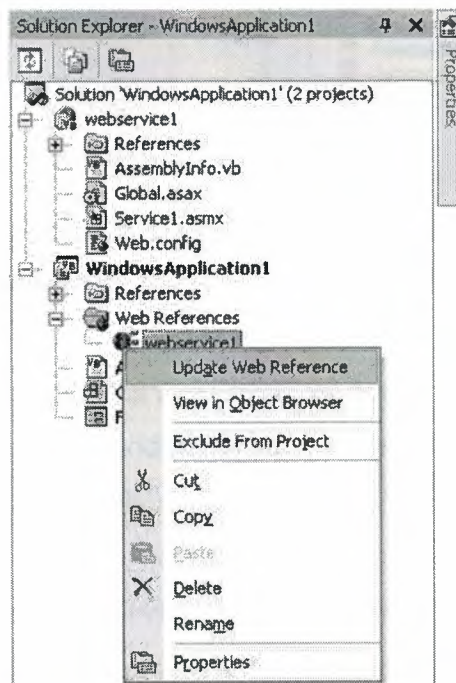


Figure 6.3.5 Updating Web Service in Windows Application1 Project

6.4 How to Write Web Service Methods?

Web Service Methods are written in the Service1.asmx file. To open Service1.asmx file double click on the file in Solution Explorer Window or right click on Service1.asmx file and click “Open”. (See **Figure 6.4.1**)

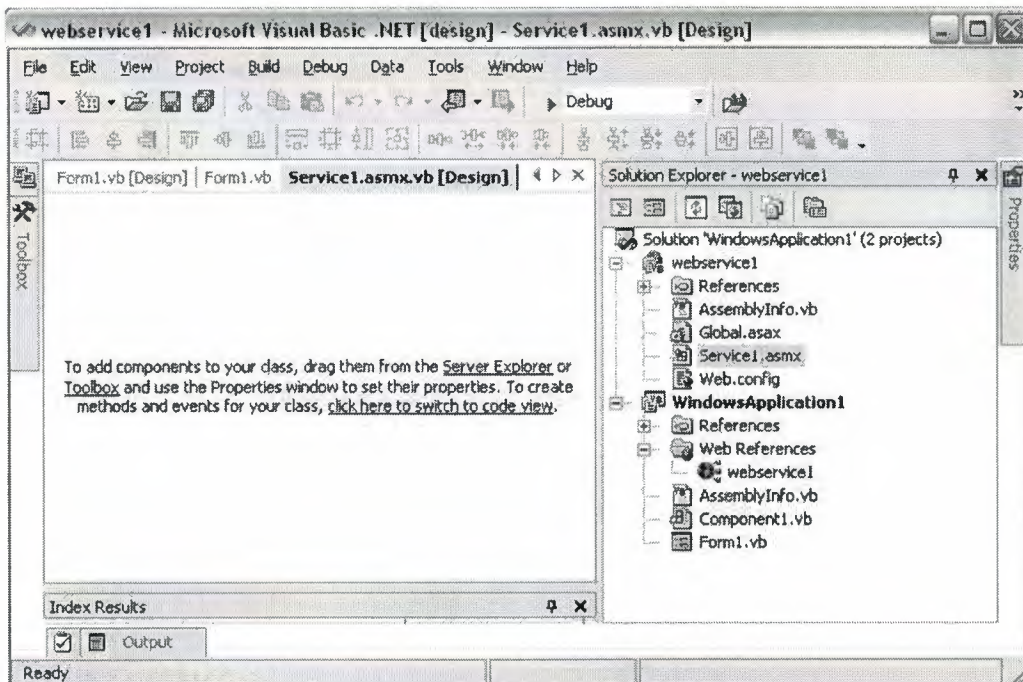


Figure 6.4.1 Service1.asmx.vb Design Window

Components can be add on to the Service1.asmx.vd design window. e.g.

OleDbConnection. To create methods and events either click the link “**click here to switch to code view**” or press **F7** shortcut. Code View is shown in **Figure 6.4.2**.

The **System.Web.Services.WebService** class is imported, which defines the optional base class for XML Web services, provides direct access to common ASP.NET objects. Each XML Web service requires a unique namespace, which makes it possible for client applications to differentiate among XML Web services that might use the same method name. The default namespace for XML Web services created in Visual Studio .NET is "http://tempuri.org/WebService1/Service1" where WebService1 is the project name and Service1 is the class name.

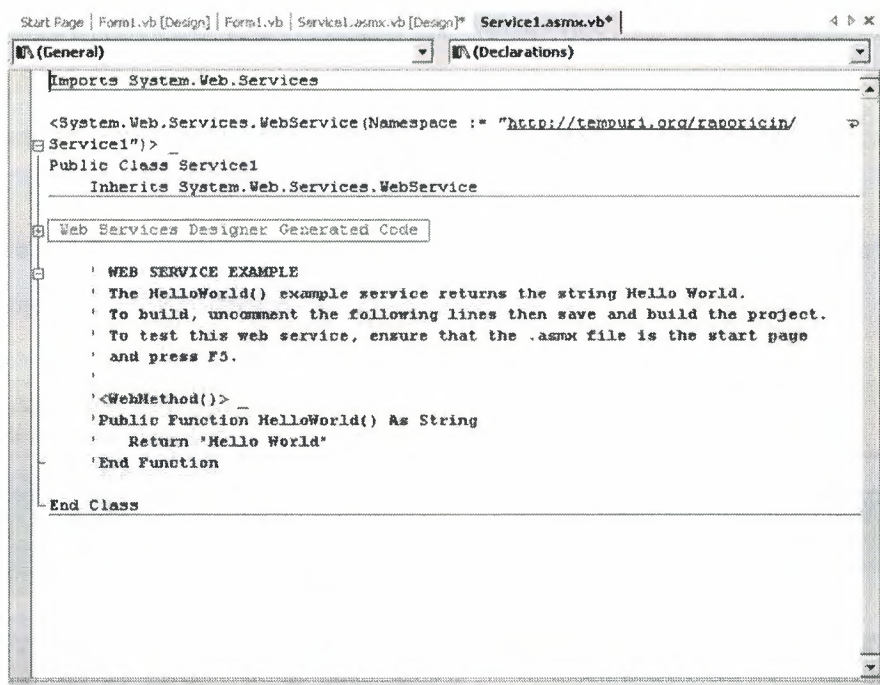


Figure 6.4.2 Code View of Service1.asmx file

Code View comes with a simple example named “HelloWorld” lies in comment as shown below:

```
'<WebMethod()> _
'Public Function HelloWorld() As String
'    Return "Hello World"
'End Function
```

The syntax for a web service method is shown below:

```
<WebMethod()>_
Public Function Func_Name(Param_name As DataType) As ReturnType
'Statements here
End Function
```

Function name and parameters name should be meaningful since the methods will be called from either web application or windows application. ReturnType should be decided carefully and when it is necessary return error messages.

When the web service is build or rebuild the Webservice1.dll file created in the bin directory of the virtual directory in the localhost. Therefore WebService1.dll is the output file for the web service project. And it will work without project files.

The **Figure 6.4.3** shows the relationship between the project, the class, its methods, and the resulting XML Web service.

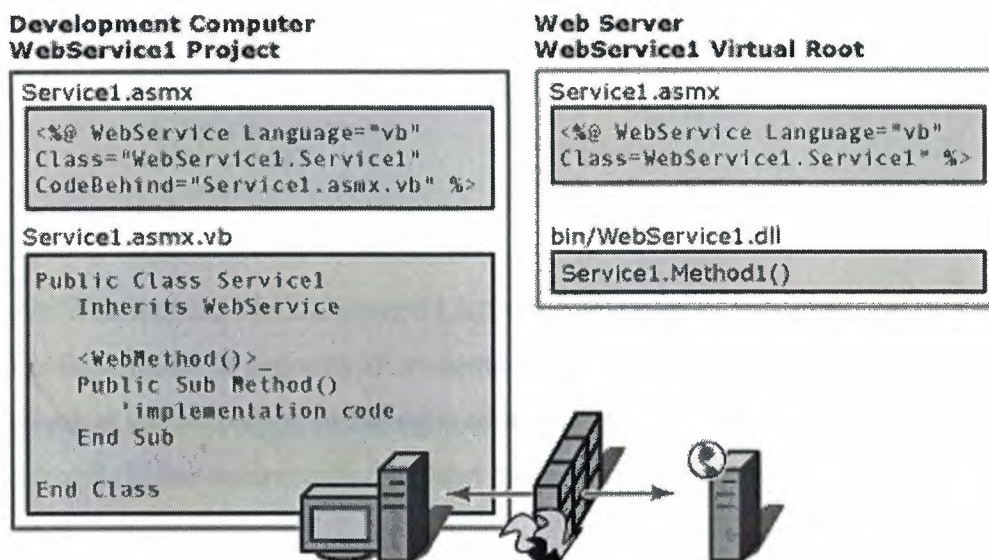


Figure 6.4.3

A simple example for Web service method

```
<System.Web.Services.WebService( _
    Namespace:=" http://tempuri.org/ConvertTemp/Service1", _
    Description:="A temperature conversion service.")> _
Public Class Service1
    <WebMethod()> _
    Public Function CTemperature(ByVal dFahrenheit As Double) _ As Double
        Return ((dFahrenheit - 32) * 5) / 9
    End Function
```

Previous web service method can be called in below format in windows forms applications or in web applications:

```
Dim myservice As New ConvertTemp.Service1      'Declare webservice
Msgbox(Myservice.CTemperature(100))           'Show Result in MessageBox
```


Actually the real use of webservice is to build distributed systems. Below code is an good distributed database example.

```
<WebMethod(> _
Public Function ListStudents(ByVal DID As Integer) As DataSet
    Dim myadap As New OleDbDataAdapter("SELECT Student.SNo,
Student.name, Student.surname, Student.sex, Student.Address,
City.CityName, Student.postcode, Student.Gsm, Student.OsymNo,
Student.phone, Student.Email, Advisor.AdvisorName, Student.StudentId
FROM Advisor INNER JOIN (City INNER JOIN Student ON City.CityId =
Student.CityId) ON Advisor.AdvisorId = Student.AdvisorId where
Student.isDeleted=false and Student.DepartmentId=@DID;",
OleDbConnection1)
    myadap.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadap.Fill(ds)
    Return (ds)
End Function
```

In above Web Service Method named ListStudents requires a parameter called DID which is department Id and lists all students in that department who are studying still. As we look at the returntype of the method it is DataSet. DataAdapter named myadap executes and fill the dataset called ds and returns the dataset.

CHAPTER SEVEN

7. ASP.NET

7.1 Introduction to Asp.NET

Most developers today are building Web applications. For the past three and half years or so, Microsoft developers have been building Web applications using Active Server Pages, or ASP. ASP is a technology in which the pages are a mix of HTML and a scripting language, such as VBScript or JavaScript. The HTML was basically static, and was rendered as you typed it in the page. The script was interpreted on-the-fly, and generated additional HTML. This generated HTML was mixed in with the static HTML, and the page was sent to the browser.

Web applications, including Active Server Pages applications, follow a simple request/response metaphor because that is all that is allowed by HTTP. The user requests a page, and the page is sent to the browser to be rendered. The person can fill out data fields, and when he clicks a button, he is making a new request, and the response is generated on the server and returned.

ASP.NET has to use request/response, of course, because you're still using HTTP. However, ASP.NET seeks to simplify the coding model, by making it appear as an eventdriven programming model.

7.2 ASP.NET Advantages Over ASP

- Eliminates long code blocks. Script is no longer intermixed with HTML. This makes the code much smaller, cleaner, and easier to maintain. This is made possible by the event-driven page processing.
- New controls have been introduced that promote user interface encapsulation. These controls give browser-independent rendering, which means that you write code only once for multiple clients.

- Page services have been introduced that reduce the grunt work involved in creating form pages that post back to themselves: **ViewState** and **PostBack** data processing.
- New application services make applications faster and more scalable. These include caching, farmable session state, and security to name a few.

7.3 How ASP.NET Works?

Basically, ASP.NET works by using server-based components to generate HTML. The HTML is sent to the client and rendered in a browser. ASP.NET determines the capabilities of the client browser and generates HTML appropriate for that browser. ASP.NET works by using server-based components to generate markup, such as HTML, and script. The HTML and script are sent to the client and rendered in a browser. ASP.NET determines the capabilities of the client browser and renders HTML appropriate for that browser. The type of markup sent to the client is determined by the controls. The markup code doesn't have to be HTML.

ASP.NET user code is precompiled. This is in contrast to ASP, which interprets the script code that is intermingled with static HTML. Even if you were using compiled COM components with ASP, the calls to the components were late-bound. Using ASP.NET allows you to benefit from all the services of the .NET Framework, such as inheritance, security, and garbage collection.

ASP.NET also provides some of the functionality that has been coded by hand in the past. Like ASP, ASP.NET can provide automatic state management. Because HTTP is a stateless protocol, maintaining state in Web applications has always been a problem. ASP.NET provides state management that, unlike ASP, is scalable across Web farms, survives IIS crashes, and does not have to use cookies.

7.4 Web Pages and Code



The pages are created in ASP.NET are divided into two parts: the **user interface** and the **code**. When you create the page in VB.NET, you see the **.ASPX** and the **.VB** files as two views of the same page. The VB file is a class file, called a page class, and it segregates your code from the HTML. When you compile the page, ASP.NET generates a new class and compiles it. This new class has the static HTML, ASP.NET server controls, and code from your form compiled in. Unlike ASP, all the HTML sent to the client is generated from the class on-the-fly. This class is actually an executable program, and whenever the page is called, the executable generates the HTML that is sent to the browser.

When a web page created, a compiled class is created from these two files. When someone browses the ASPX page, the class is executed and generates the HTML to send to the browser. HTML sent to the browser, has **<FORM>...</FORM>** block and all the controls added on the form should be in these tags. This is because an HTML form is the only way for standard HTML to get data from an HTML page back to the server.

The code written for an event runs only on the server. When someone calls the event, and the form is submitted to the server. In effect, you take the event and send it to the server for processing. The generated class is instantiated, and the event code is processed. A new HTML stream is generated and sent back to the client browser.

7.5 Basic Web Controls

Server Control	Characteristics
Label	A Label is used to display text. If we want to display static text, we do not need a Label server control; we should instead use HTML. We should use a Label server control only if we need to change its properties via server code.
TextBox	A TextBox control enables the user to enter text. By default, the TextMode property is SingleLine, but it can also be set to Multiline or Password. In case of Multiline text box, the Rows property determines the height. If its AutoPostBack property is set to True, it generates a PostBack on its Text_Changed() event.
Buttons:	All three types of buttons cause PostBacks when the user clicks them.
■ Button	Button controls can be placed inside other container controls, such as DataList, DataGrid and Repeater.
■ LinkButton	The LinkButton renders a hyperlink in the page.
■ ImageButton	The ImageButton displays an image that responds to mouse clicks. We can also use it as an image map. Thus, we may pinpoint where in the graphic the user has clicked.
CheckBox	It enables the user to input Boolean data: true or false, yes or no. Its Checked property can also be bound to a data field of a data source. Its CheckedChanged event can be used for AutoPostBack.
ListControls: ■ CheckBoxList ■ DropDownList ■ ListBox ■ RadioButtonList	These controls are derived from the ListControl abstract class. Note: these controls will be discussed in detail in a later section of this chapter.
HyperLink	It displays a link to another page. It is typically displayed as text specified in its Text property. It can also be displayed as an image specified in the ImageUrl property. If both the Text and ImageUrl properties are set, the ImageUrl property is displayed. If the image does not exist, then the text in the Text property is shown. Internet Explorer uses the Text property to display ToolTip.
Image	We may use the Image control to display an image on the Web page. The ImageUrl property specifies the path to the displayed image. When the image does not exist, we can specify the text to display in place of the image by setting the AlternateText property. The Image control only displays an image. If we need to capture mouse clicks on the image, we should instead use the ImageButton control.

Server Control	Characteristics
Panel	This can be used as a container of other controls. This control is rendered as an HTML <div> element.
RadioButton	It creates an individual radio button on the page. We can group them to present mutually exclusive choices.
Table	It enables us an HTML table. A table can be built at design time with static content, but the Table control is often built programmatically with dynamic contents. Programmatic additions or modifications to a table row or cell do not persist on PostBack. Changes to table rows or cells must be reconstructed after each post to the server. In these cases, better alternatives are DataList or DataGrid controls.
Xml	This control can be used to transform XML documents.

Many of the basic server controls work very similarly to their HTML server control counterparts. All of the Web controls are prefixed with *asp:* in their tags. For example, the tag for a label Web control is `<asp:Label>`. Their uses are also mostly intuitive. All of the examples illustrated in the HTML server control section can also be effectively developed using Web controls.

7.6 First ASP.NET Application

Start Visual Studio.NET and choose to create a new Visual Basic project using the Web Application project and name the project WebAppTest. Notice, as shown in **Figure 7.6.1**, that the location of the project is an HTTP address, not a directory on the machine. The server to which you connect must have **Internet Information Server (IIS)** 4.0 or higher. The server must also have the .NET Framework loaded, so you might want to use your local machine as the Web server.

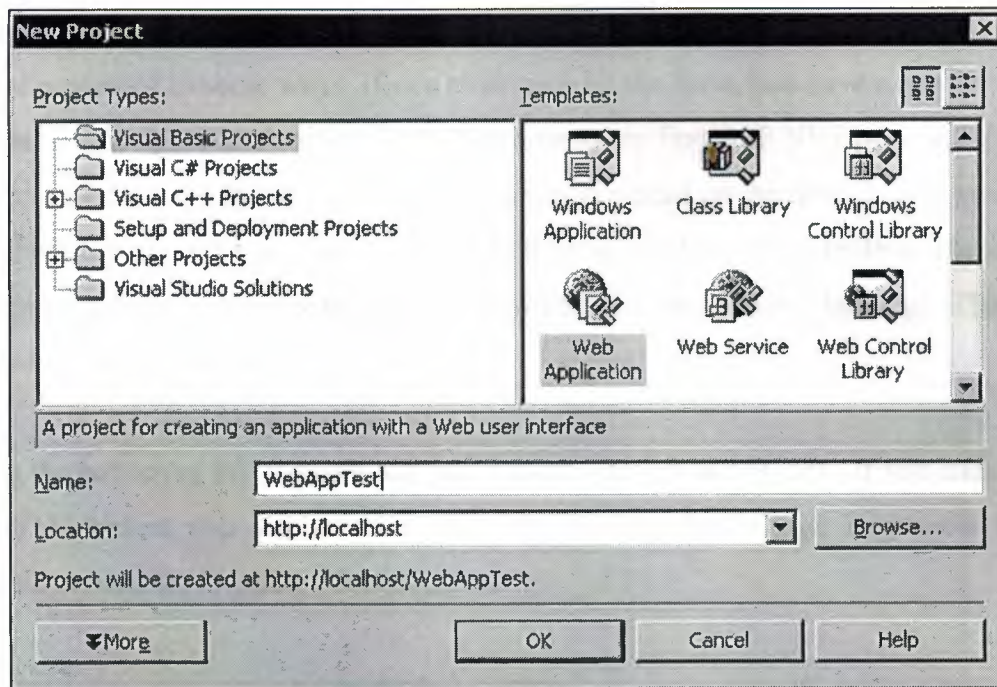


Figure 7.6.1

After you click the OK button, VS.NET attempts to communicate with the Web server. Provided this communication is successful, the project is created on the Web server, and you are ready to begin working with the project.

The page will open as a blank form in the designer, with a little descriptive text in the middle. If you look in the Solution Explorer, you will see that this page is named WebForm1.aspx. ASP uses the .asp extension, whereas ASP.NET files use the .aspx extension. ASP and ASP.NET can coexist in the same directory if necessary. If you look at the editor, it is just a blank form right now. The default view is for the form to be in GridLayout mode, which means that you can drag and drop controls onto the form and easily position them by using the standard snap-to-grid feature of the designer. There is also a FlowLayout mode that allows you to get absolute positioning by placing the controls exactly where you want. To switch between FlowLayout and GridLayout modes, change the pageLayout property in the Properties window.

Go ahead and switch the `pageLayout` property to `FlowLayout`. This mode works like a word processor in some ways. If you click once on the form, you have a cursor blinking in the upper-left corner. Type `Welcome to my first ASP.NET page` and then press the Enter key. As you can see, the text is placed on the form, just as you would expect in a word processor. Highlight the text and look at the toolbar. There is a drop-down box that says `Normal`. Drop down this list and choose `Heading1`. The text enlarges significantly.

Along the bottom of this window are two buttons: `Design` and `HTML`. If you click the `HTML` button, you will be shown the HTML making up the page. Right now, the line that creates the `Heading1` is as follows:

```
<H1>Welcome to my first ASP.NET page</H1>
```

Now, go back to the `Design` view by clicking the `Design` button. Highlight the text again, and click the `Center` button to center the text. If you switch back to the `HTML` view, you will see that the earlier `Heading1` line has changed to this:

```
<H1 align=center>Welcome to my first ASP.NET page</H1>
```

At this point, you might think you have a high-powered HTML editor, not much different from `FrontPage` or a hundred other HTML editors. Now, however, it is time to see an example of some ASP.NET. Go back to the `Design` view and move to the line below the `Heading1` line you added. Click on the `Toolbox` and notice that there is a tab for `Web Forms`. (See **Figure 7.6.2**)

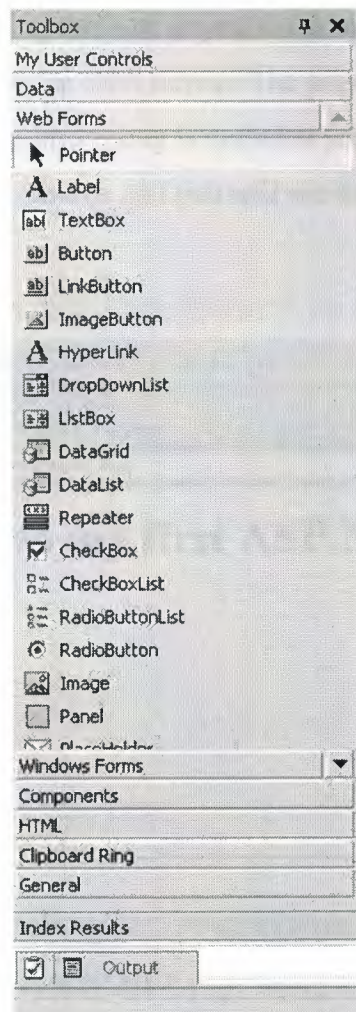


Figure 7.6.2 Web Form Tab in Toolbox

Next, click and drag a button to the form. You should now have a label and a button next to each other. Double-click on the button and you will open the code window. Notice that this code window creates a file with the same name as the ASPX, but the file has a .VB extension. As you will see, this is called a **code-behind page**. One of ASP.NET's design goals is to separate the code and the user interface.

In the code window, you will be in the **Button1_Click** event procedure. Type the following code:

```
Label1.Text = "Hello, World!"
```


Notice that you are programming this just as you would a standard Windows application. Go ahead and click the Start button. The page renders in the browser, as shown in **Figure 7.6.3**. What is interesting is not what you see in the browser, but the code behind it. Click on View, Source and you will see the HTML that is making up the page.

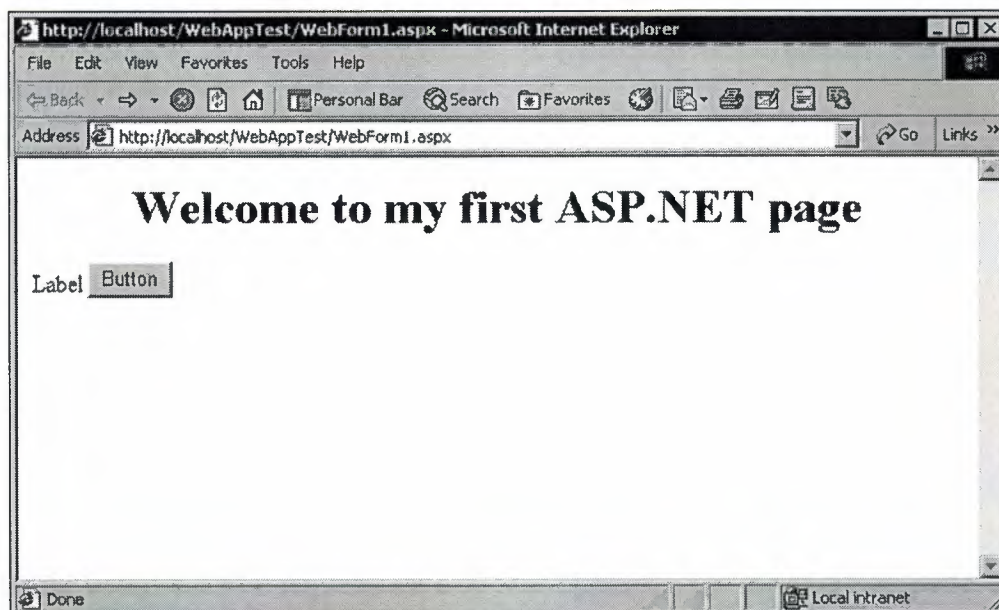


Figure 7.6.3 Your first ASP.NET page being rendered in the browser.

HTML code is below:

```
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft Visual Studio.NET 7.0">
<meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
<meta name="vs_defaultClientScript" content="JScript">
<meta name="vs_targetSchema" content="Internet Explorer 5.0">
</HEAD>
<body>
<form name="WebForm1" method="post"
action="WebForm1.aspx" id="WebForm1">
<input type="hidden" name="__VIEWSTATE"
value="dDwtMzQ3NzI5OTM4Ozs+" />
<H1 align=center>Welcome to my first ASP.NET page</H1>
<P align=left>
<span id="Label1">Label</span>
<input type="submit" name="Button1" value="Button" id="Button1" />
</P>
</form>
</body>
</HTML>
```

Notice that there is no VB code here. You created a procedure for the Button1_Click event and typed a line of code. However, none of that has made it to the client. This is because ASP.NET just sends HTML to the client, whereas the code is compiled and lives on the server. That means this page can be used by anyone, using any browser and operating system.

Test the page by clicking the button. You will notice the text Hello, World! Now appears in the label. If you click View, Source again, you will notice that the Label1 tag has changed from this:

```
<span id="Label1">Label</span>
```

to this:

```
<span id="Label1">Hello, World!</span>
```

7.7 Validation Controls

One of the most common requests for any Web application is the ability to perform client-side validation of input. In standard HTML, there is no way to perform validation of data on the client. To get around this problem, most browsers let you mix in some client-side script code, which is capable of performing validation, but the code for this can be tedious to write. There are a number of reasons for performing validation on the client. First, you give the user a better experience. If you can immediately notify the user that he did not fill in a required field, you just saved him the time it would have taken to submit the form, have the server generate a message to inform him of the problem, and return the error message to him.

Microsoft provides a series of validation controls in ASP.NET that automate client-side form validation. The validation controls in ASP.NET are smart; they will perform the validation at the client if possible. If the client can handle DHTML, the validation controls send the code down to the client. If the browser is less capable, the validation code is actually executed on the server.

Add a new Web Form to your project and name it `UserInfo.aspx`. Change `UserInfo`'s `pageLayout` property to `FlowLayout`. Click on the Toolbox and drag a text box from the Web Forms tab and drop it on your new Web form. Now, on the form, click to the right of the text box so that you can see the cursor. Press the Enter key to move to the next line. Click on the Toolbox, drag a button from the Web Forms tab, and drop it on the line below the text box. Now, from the Web Forms tab of the Toolbox, drag a `RequiredFieldValidator` control and drop it next to the text box. The `RequiredFieldValidator` control checks whether a particular field has been filled in. You place the `RequiredFieldValidator` on the page, and then tie it to a particular input control, such as a `TextBox`, `CheckBox`, or `DropDownList`. In this case, you want to tie it to the text box. Click once on the `RequiredFieldValidator` if it is not already the current object. In the Properties window, you will see a property named `ControlToValidate`. Click here and drop down the list. The only input control that will appear is `TextBox1` choose it.

It is required to go to the project properties and choose to make `UserInfo.aspx` the startup object. However, Web Applications work differently: There is no startup object. Right-click on `UserInfo.aspx` in the Solution Explorer window and choose `Set As Start Page`. This notifies Visual Studio.NET which page to run when the user clicks the Start button. It does not change anything in the page itself. Now that you have set `UserInfo.aspx` as the start page, run the project. The page appears inside Internet Explorer.

After the page is running, click on the button without entering anything in the text box. Immediately, the message `RequiredFieldValidator` appears to the right of the text box as shown in **Figure 7.7.1**. Now, enter anything into the textbox and click the button. The `RequiredFieldValidator` text goes away, and the entered value stays in the text box. When the `RequiredFieldValidator` text disappears and the value you typed in the text box remains, it means that the submit action has taken place, which means you have made a round trip to the server.

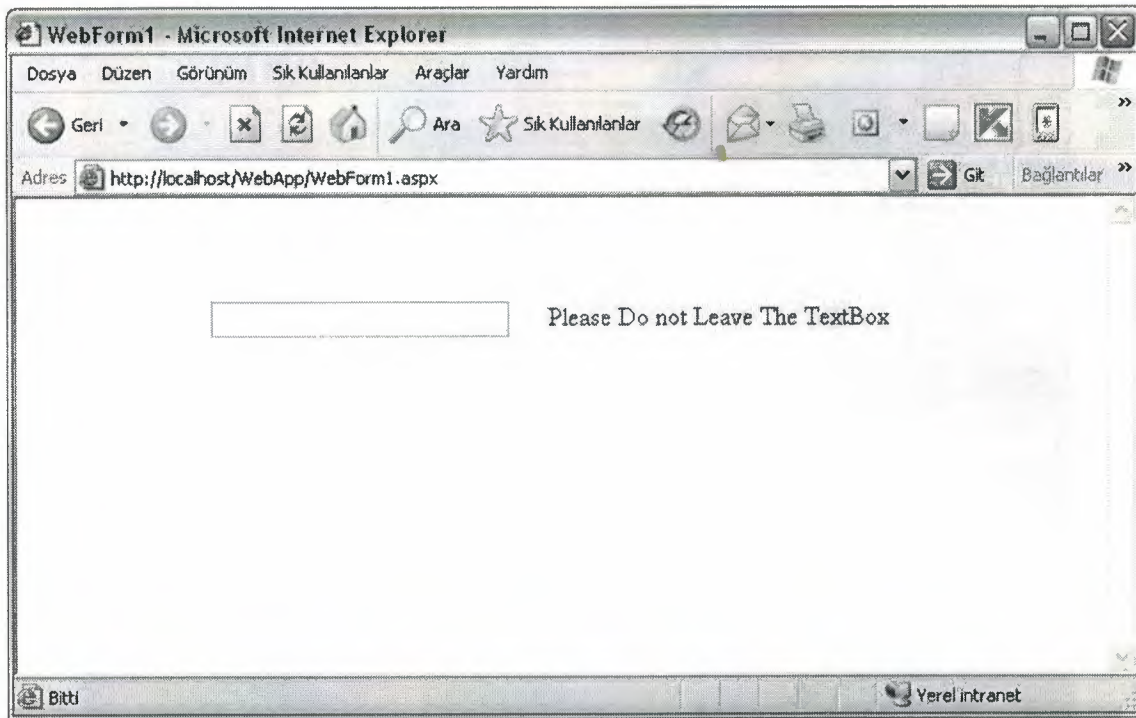
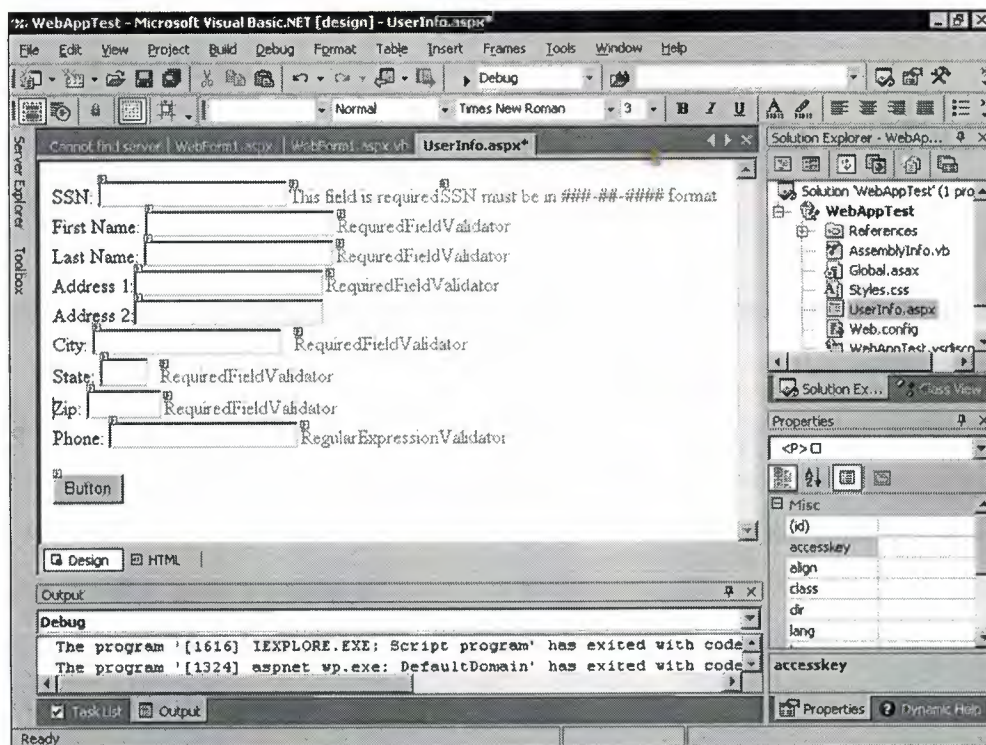


Figure 7.7.1 RequiredFieldValidator in ASP.NET

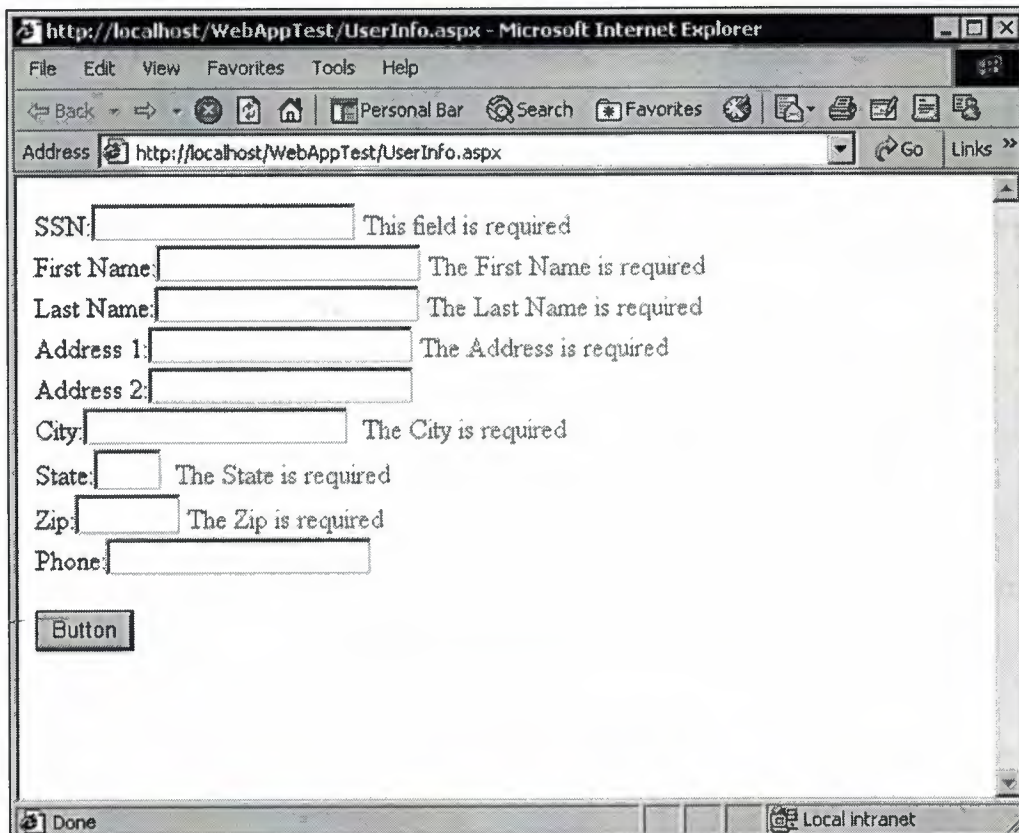
If you use IE 4.0 or higher, the validation actually occurs on the client. This allows you to get immediate feedback that the field is blank, when in fact you specified that a value is required. After you fill in the value, you send the data to the server and perform a server round trip.

The error message is default as the name of the validator. For example the default error message for RequiredFieldValidator is "RequiredFieldValidator". To make sense the error messages should be meaningful. All validators have ErrorMessage property.

A more complex data entry screen with validator server controls is shown in **Figure 7.7.2** with default Error messages and with modified ErrorMessage in **Figure 7.7.3**



Figures 7.7.2 Validators with Default ErrorMessages



Figures 7.7.3 Validators with Modified Messages

7.8 Types of Validators

If you look at the Web Forms tab of the Toolbox, you'll see a variety of validator controls. Briefly, they are as follows:

- **RequiredFieldValidator:** This validator requires that its `ControlToValidate` property have a value. In other words, the control to which this validator is tied cannot be left blank.
- **CompareValidator:** This validator compares the value the user entered with a value you specify. Your specified value could be a constant, a calculated value, or a value from a database.
- **RangeValidator:** This validator requires that entered data be within a particular range. The range can be numeric, dates, currency, or alphabetical.
- **RegularExpressionValidator:** Regular expressions are also known as masks. This validator can make sure that entered data matches a particular format, such as the format of phone numbers and Social Security numbers.
- **CustomValidator:** This validator uses code you write yourself to validate the data.
- **ValidationSummary:** This validator simply reports all the errors encountered by the other validators. You will see an example of this validator shortly.

There are a number of common properties in these controls. The major ones are:

- **ErrorMessage** In case of an error, the system displays this message at the location of the control, and in the summary report, if any.
- **Display** A validation control is kept invisible until a bad input is entered. In case of a bad input, the system has to display the error message. The display mechanism can be handled in one of three ways.
 - i. **Display= "static"** Initially, enough room in the page is reserved for the expected error message.
 - ii. **Display= "dynamic"** No room is initially reserved. In case of an error, the message is displayed by displacing existing contents of the page.
 - iii. **Display="none"** The message won't be displayed at the location of the control; however, it will be reported in the summary report, if any.

7.9 The Databound ListControls Family

This family of controls is new to ASP developers. These controls provide rapid application development to display and manipulate data from any data source. The following controls shown in **Figure 7.9.1** belong to this family.

CheckBoxList	DataGrid	DataList	DropDownList
HtmlSelect	ListBox	RadioButtonList	Repeater

Figure 7.9.1 The Databound ListControls Family

Data binding means binding controls to information stored in a data store. Here, the term "data" is used in a very broad sense. When we talk about data binding, it implies binding any control property to almost any kind of data store. A data store can be as simple as a public property on a page, or as complex as a database stored on a server. This broad choice among data stores provides high flexibility, and thus enables you to bind a control to any data store based on your need. The good news about data binding is that many of the same data controls are used in ADO.NET.

The Web Forms controls that are bound to a data store access data through the properties of specific classes, categorized as data classes. Data classes typically include methods that can be used for updating the underlying data stores.

You can bind a control to different data stores, such as properties, methods, or collections. These different data stores can be bound to a control property by using data binding expressions. While binding, the data is always bound to a particular property of the control (the property name might differ for various controls). When a data binding expression is evaluated, the resulting value is loaded in the control's bound property.

When you bind a control property to a data store, the Web Forms Framework cannot evaluate data binding expressions automatically. To display the evaluated value in the control's bound property, you need to call the **DataBind() method** explicitly. The page and each control on the page support this method. When you call the DataBind() method for a control, it is cascaded to all its child controls.

7.9.1 Using the Databound ListControls

i. DropDownList Control

DropDownList control can be used to display bound data. The advantage of the DropDownList in ASP.NET to other languages is that DropDownList holds two fields , one which is the text the user sees, other which is the value that is not shown to the user.

Some properties of the DropDownList Control is listed below:

DataTextField: Field that is shown to the user binded to a specific datafield in datasource .

DataValueField: Field that is not shown to the user binded to a specific datafield in datasource . Generally the primary key is contained in this field.

```
Sub bindListControl()  
Dim myadapter As New OleDb.OleDbDataAdapter("Select cityname,cityId  
from city", OleDbConnection1)  
Dim ds As New DataSet  
myadapter.Fill(ds)  
citylist.DataSource = ds.Tables(0)  
citylist.DataTextField = "cityname"  
citylist.DataValueField = "cityId"  
citylist.DataBind()  
End Sub
```

The result page of the above Code is shown in **Figure 7.9.1.1**.

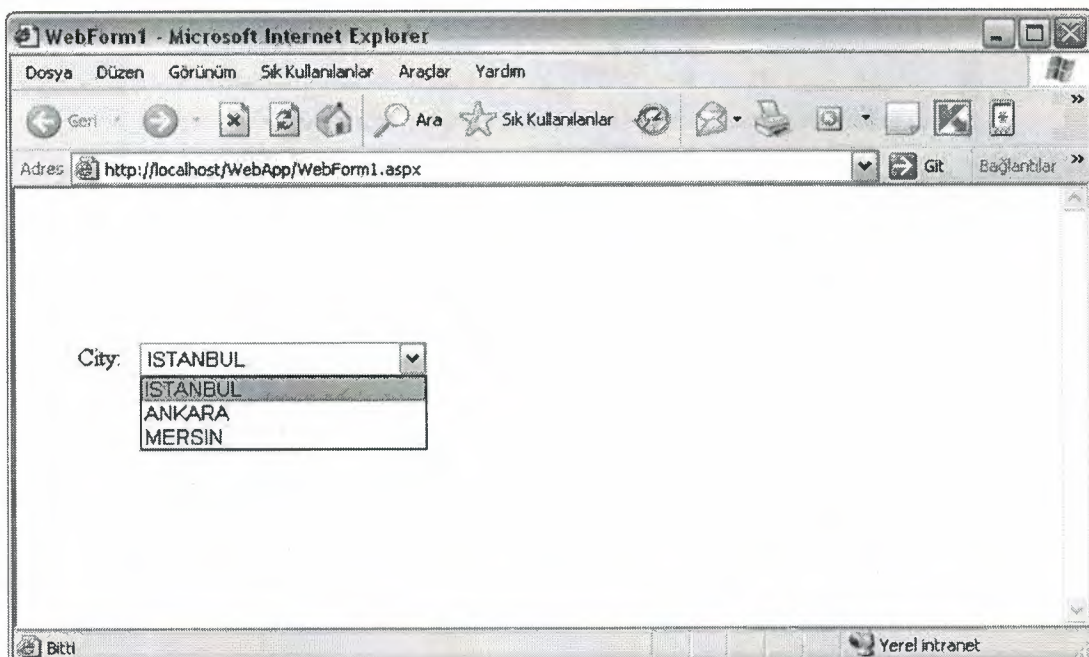


Figure 7.9.1.1 Binded DropDownList

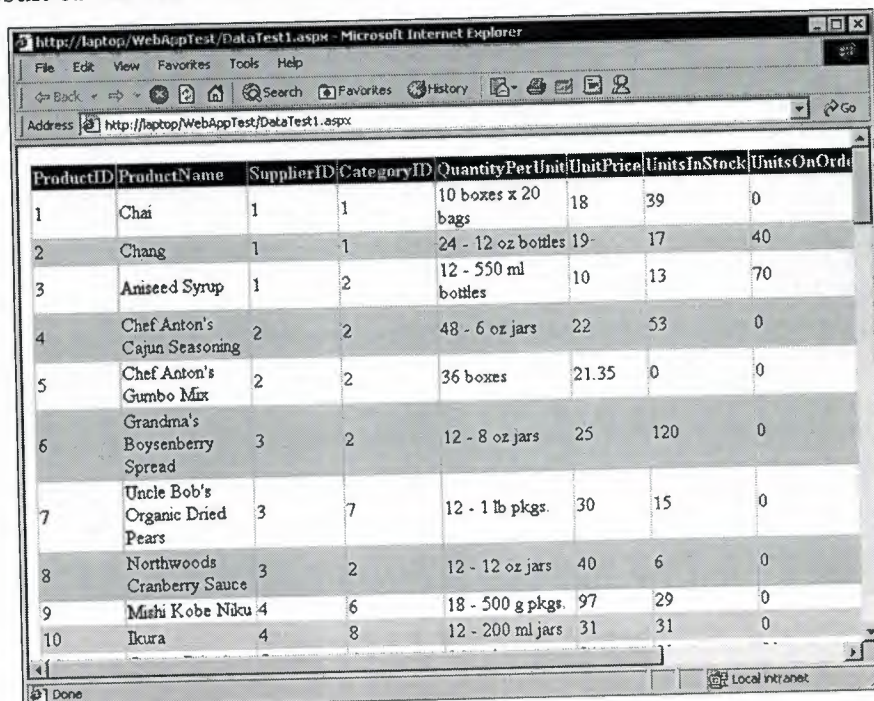
ii. Data Grid Control

The DataGrid Control happens to be the most powerful member of the data-bound control family. DataGrid control offers **sorting** and **paging capabilities**. We can employ its <AllowSorting> property to dynamically sort and re-display data on selection of a column header. In case of very large data source, we can use its <Allow Paging> property to display a selected page of data.

Essentially, a DataGrid control can be used to display bound data in tabular format. Each record in the data source is displayed as a row in the grid. By default, the data grid maps each field of the data source as a column in the grid. Obviously, we may override the default value of its AutoGenerateColumn property to display selected columns in a particular order.

```
Sub bindListControl()  
Dim mydataset As New Dataset  
sqlStr=" Select * from Products"  
myOleDbAdapter =New OleDbDataAdapter(sqlStr,myConn)  
myOleDbAdapter.Fill(myDataSet,"dtProducts")  
DataGrid1.DataSource=myDataSet.Tables("dtProducts")  
DataGrid1.DataBind()  
End Sub
```

The result of the above sub routine is shown in **Figure 7.9.1.2**.



The screenshot shows a Microsoft Internet Explorer window with the address bar displaying 'http://laptop/WebAppTest/DataTest1.aspx'. The main content area displays a DataGrid control with 10 rows of product data. The columns are ProductID, ProductName, SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, and UnitsOnOrder. The data is as follows:

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder
1	Chai	1	1	10 boxes x 20 bags	18	39	0
2	Chang	1	1	24 - 12 oz bottles	19	17	40
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10	13	70
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22	53	0
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35	0	0
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25	120	0
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30	15	0
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40	6	0
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97	29	0
10	Ikura	4	8	12 - 200 ml jars	31	31	0

Figure 7.9.1.2 Binded DataGrid

CHAPTER EIGHT

8. EDUCATIONAL ORGANIZATION SOFTWARE

8.1 How Educational Organization Software Works?

The project is on Education Software. The software can work at anywhere where an internet connection is established. Hence it is a powerful program for the educational organizations which has several branches in different places. The organization will use only one windows program that will be on local on each computer and the database will be on the HTTP. Hence all datas are in one database file and program will communicate with HTTP to work with those datas.

Figure 8.1.1 shows the Welcome screen for the Educational Organisation Software.

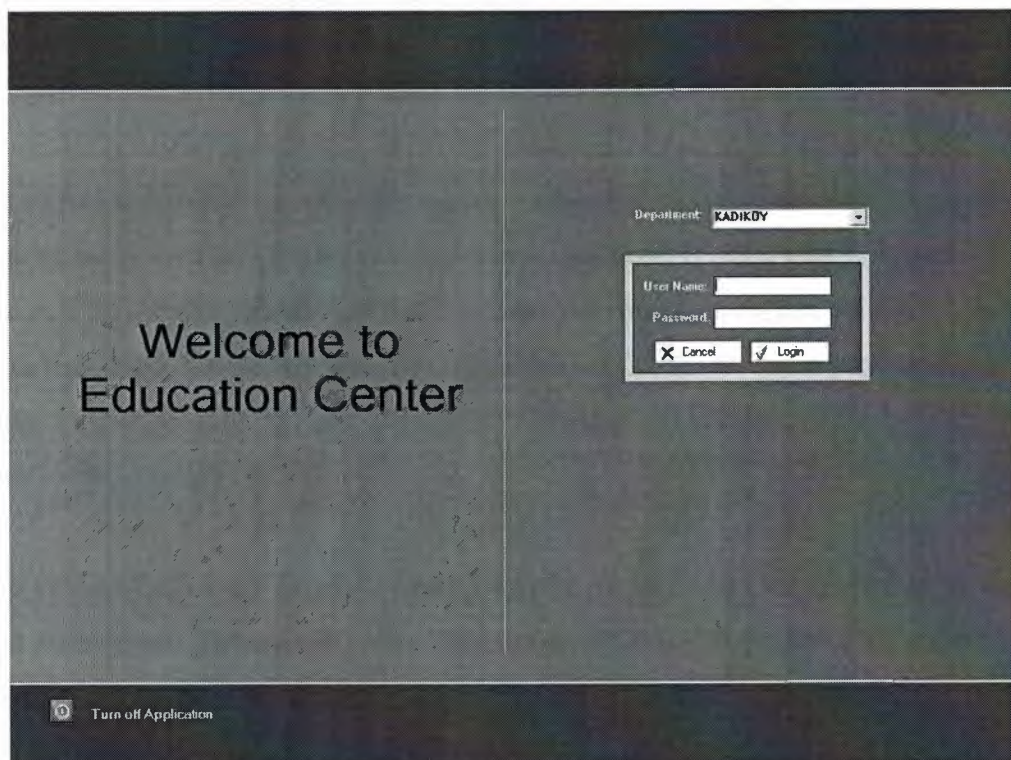


Figure 8.1.1 Welcome Screen

In Welcome Screen the software communicates with the database which is on the internet and list the Departments to the combobox.

After a registered Username and Password are written in the fields , Login button will start the main form which is shown in **Figure 8.1.2**.

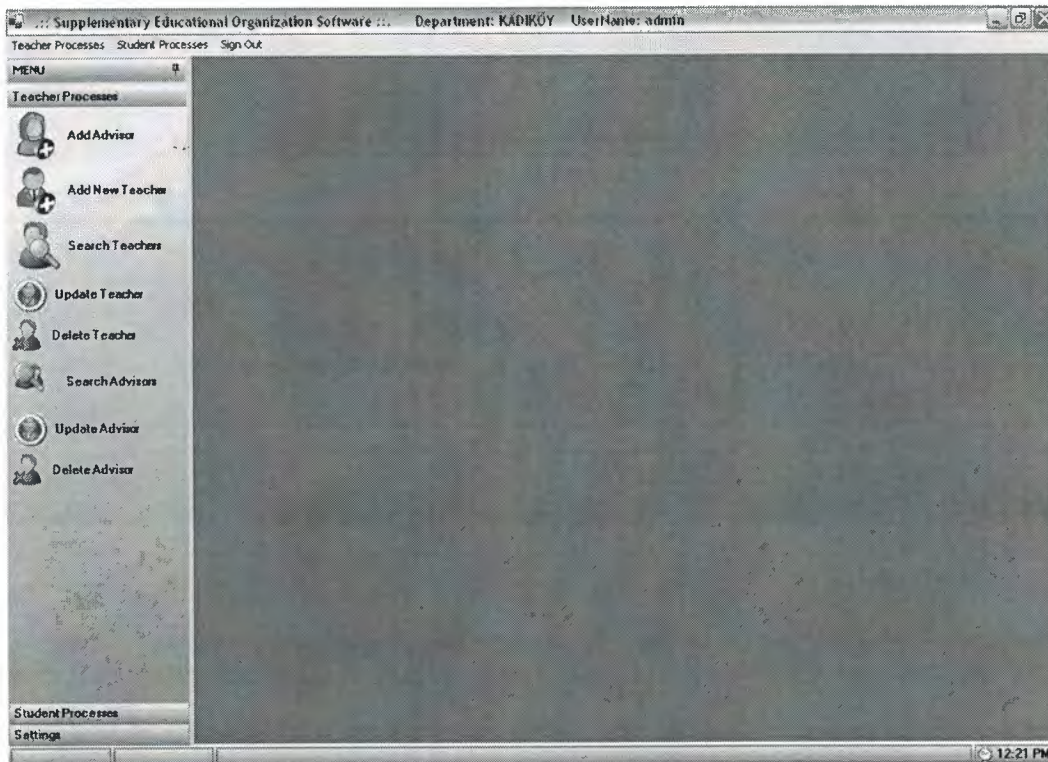


Figure 8.1.2 Main Form

Main form has a mdi Form property which contains child forms that will be created as buttons click events called. In left side a menu is appear. It has 3 tabs Teacher Processes, Student Processes and Settings tabs. As shown in **Figure 8.1.2** the teacher tab contains Add, Delete, Update, Search processes. Those are main processes that can be done to both teacher, student and advisor. Settings tab is a special tab because it is shown to the admin users only.

In top of the window the software name, department name and the username is written as the form opens. There is a clock at the bottom right of the window on statusbar that takes the system time.

It is not possible to explain and show all forms since there are 38. A brief explanation for several forms will be done. Now lets see the Add New Teacher Form in

Figure 8.1.3

Figure 8.1.3 Add New Teacher Form

As it is said before the Main Form is mdi parent and all the other forms that will be opened will be mdi child of main form. Teacher informations are taken in that form and new teacher is added. Notice that there are some icons besides the textboxes. These icons are ErrorProviders which avoid wrong or missing entries. For example Name field is empty and lost the focus then as you point the mouse on the icon and wait for a second the error message will appear saying that 'Do not Leave Name Field Empty'. Since the errors shown before user presses the buttons, the time is saved from pressing the button and then showing the error. This will save time for user.

No messagebox is used in that software. MessageBox will appear and wait user to press OK button to disappear and sometimes they can be disturbing when appears after every operation. A userdefined Notifier is used in that project. This will show an Title with message and disappear in 2 seconds or just by clicking the 'x' button on it. The **Figure 8.1.4** shows an example view of the notifier.

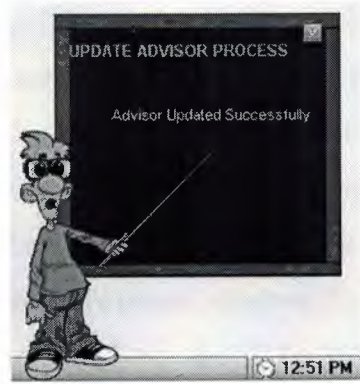


Figure 8.1.4 Notifier after Update Process

Lets see the Search Teachers Form in **Figure 8.1.5**.

Supplementary Educational Organization Software : Department: KADIKÖY Username: admin - [List Teachers]

Teacher Processes Student Processes Sign Out

MENU

Teacher Processes

Add Advisor

Add New Teacher

Search Teachers

Update Teacher

Delete Teacher

Search Advisors

Update Advisor

Delete Advisor

Student Processes

Settings

All Teachers Email Unregistered Deleted Other

TeacherId	Tname	Tsurname	Address	Gsm	Phone	Email	Birthdate
1	OKAN	TEKELI	başkent	0532xxxxxxx	212xxxxxxx	okan.tekeli@y	9/20/1976
2	SEVİM	MERAL	kadıköy				1/5/1970
14	ANIL	KOC	ortaköy		212 3231232		12/13/1963
15	BURHAN	KARA	besiktas				12/13/1975
17	ORHAN	ORTA	meddyoköy				12/13/1967
19	FATİH	YAR	Sarıyer				12/13/1955

Name & Surname

Class

Add Update Delete Details Send Mail Register

1:05 PM

Figure 8.1.5 Search Teachers Form

In this form All teachers are listed. And a variety of filter options are served for the user. At the top of the page the radiobuttons will filter the teachers. For example Email radiobutton will list only the teachers who have email address. And at the right top there is a radiobutton named Other. This will cause the panels at left of the screen to roll down and see the other search criterias. In that part you can search with name and surname, and by the course he/she gives.

Add, Delete , Update buttons will immediately call the forms Add New Teacher Form, Update Teacher and Delete Teacher. **Details** button will create a form which shows teacher information on a form in detail. Why it is needed is a nice question since some fields can be more than the field size on the datagrid then it will be torture to read those fields. **Send Email** button will create a form which has a similar view with an outlook window but notice that this is not an outlook window. The System.Web.Mail.SmtpMail class is used to send mails. Therefore Smtp should be installed in the user's computer. Register button will create and call a form that has a combobox which lists all the courses that are not given to any teacher. So that any listed teacher can be registered to those courses. Register Form is shown in **Figure 8.1.6**

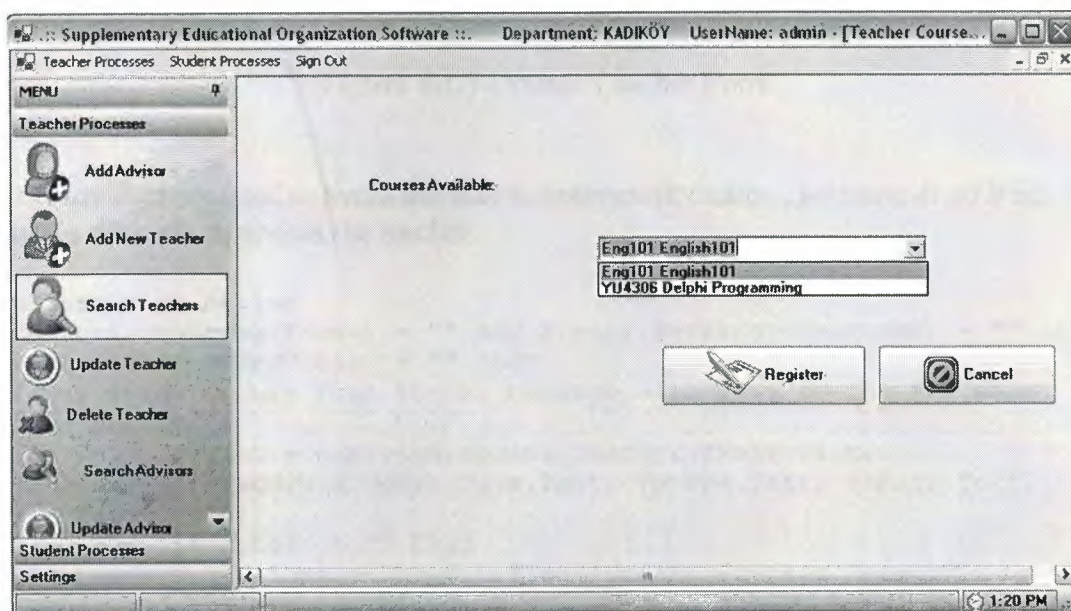


Figure 8.1.6 Teacher Course Registration Form

Lets pass to the Update Teacher Form shown in **Figure 8.1.7**. Actually to update a teacher first the teacher who will be updated, should be found so as user presses the Update Teacher button the Search Teachers Form will appear. After selecting the teacher who will be updated the Update button is pressed which will list the information of the teacher to be updated.

Figure 8.1.7 Update Teacher Form

ErrorProviders are used to avoid the user to enter empty name , surname or address. Lets see the code to update the teacher.

```
Dim result As String
If Error1.GetError(TName) = "" And Error1.GetError(Tsurname) = "" And
Error1.GetError(Taddress) = "" Then
'If any error occurs then the ex message will pass to the variable
called result
        result = wservice.Update_Teacher(TName.Text,
Tsurname.Text, Taddress.Text, Tgsm.Text, Tphone.Text, Temail.Text,
BDate.Value, TID)
        If result = "" Then
            Mainpage.ShowNotifier("Update Process Result",
"Teacher Updated Successfully")
        Else
            Mainpage.ShowNotifier("ERROR OCURED DURING UPDATE ",
result)
        End If
    End If
```

In this code block the ErrorProvider named “Error1”s GetError method used to check if there is an error and if not the **Update_Teacher()** method is called from the webservice and the required parameters are passed to that method. The returntype is string from that method as you see and it returns error if there is any. By checking the content of that result variable the proper message is shown to the user with notifier which is declared in Mainpage. The “ShowNotifier” sub routine will create and show the notifier with the given message and title. The Webservice method to update teacher is follow:

'Function to Update Teachers

```
<WebMethod()>
Public Function Update_Teacher(ByVal name As String, ByVal surname
As String, ByVal address As String, ByVal gsm As String, ByVal phone
As String, ByVal email As String, ByVal bdate As Date, ByVal TID As
Integer) As String
    Dim mycommand As New OleDb.OleDbCommand
    mycommand.Connection = OleDbConnection1
    mycommand.CommandText = "Update Teacher set Tname=@name,
Tsurname=@surname, address=@address, Gsm=@gsm, phone=@phone,
Email=@email, Birthdate=@Bdate where TeacherID=@TID"

    mycommand.Parameters.Add("@name", name)
    mycommand.Parameters.Add("@surname", surname)
    mycommand.Parameters.Add("@address", address)
    mycommand.Parameters.Add("@gsm", gsm)
    mycommand.Parameters.Add("@phone", phone)
    mycommand.Parameters.Add("@email", email)
    mycommand.Parameters.Add("@Bdate", bdate)
    mycommand.Parameters.Add("@TID", TID)

    Try
        OleDbConnection1.Open()
        mycommand.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try

End Function
```

Lets see how a teacher is deleted in **Figure 8.1.8**

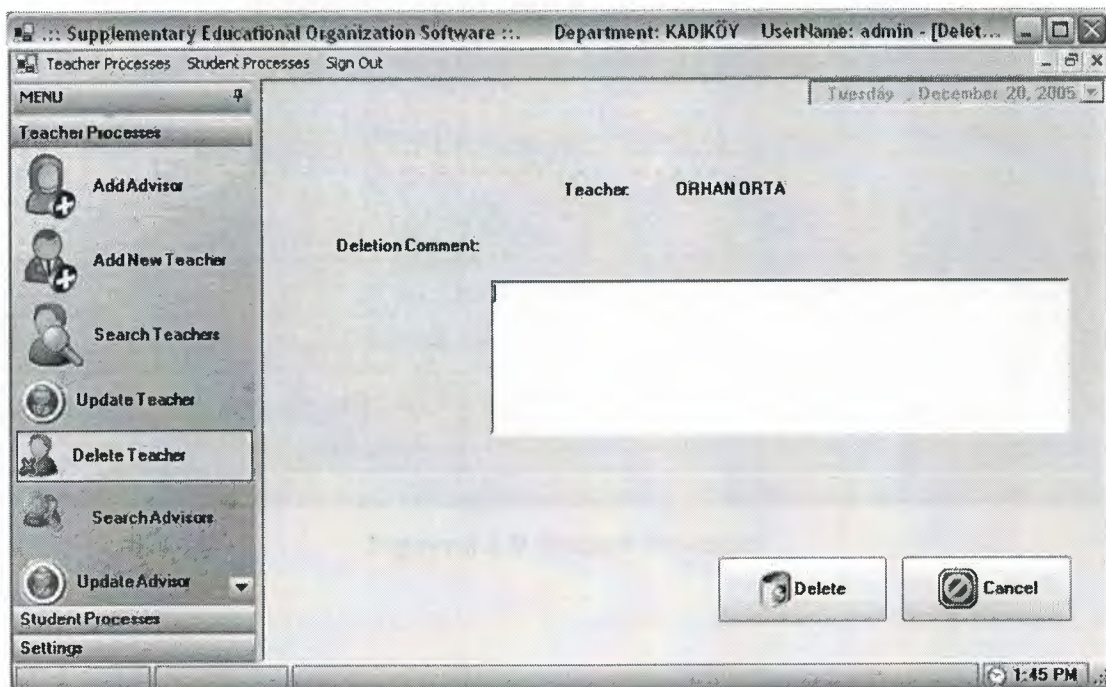


Figure 8.1.8 Delete Teacher Form

Since the teachers are related to the courses they shouldn't be deleted physically. So that a field named "**isDeleted**" in the Teacher Table in database is used to differentiate the deleted and active teachers. For deletion process a deletion comment is required which will explain why the teacher is deleted, and the deletion date which is the current date. The code for deletion process which is in delete_click event procedure is below:

```
Dim result As String
result = wservice.DelTeacher(TID, DelDate.Text, Comments.Text)
If result <> "" Then
    Mainpage.ShowNotifier("ERROR OCURED DURING DELETION ", result)
Else
    Mainpage.ShowNotifier("DELETION PROCESS ", "Teacher is Deleted Successfully")
    Me.Close()
End If
```

For Advisor process everything is same in Teacher processes. Since advisors are only deal with students the only difference is that there is no register to course process. So lets jump to the **Student Processes**. See **Figure 8.1.9**

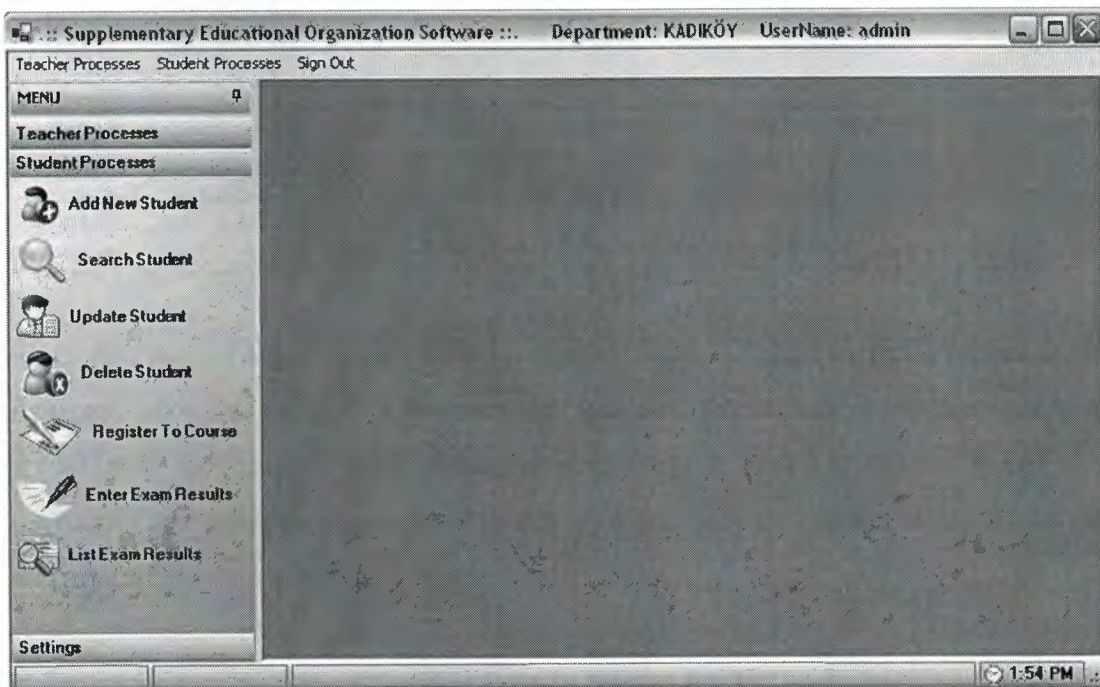


Figure 8.1.9 Student Processes

As shown in **Figure 8.1.9** the Student Processes tab contains Add, Delete, Update, Search, Register, Enter Exam Results and List Exam Results processes. Simply Add New Student button creates and call a form that has several tabs which contain fields for data entry. These tabs are Personal Informations, Contact Informations, Course Informations, Payment Informations. Moving to the other tabs can be done by holding mouse on the tab you wish , it will automatically go to that tab and show the content. See **Figure 8.1.10**.

The screenshot shows a web application window titled "Supplementary Educational Organization Software". The top bar indicates the department is "KADIKÖY" and the user is "admin". The left sidebar contains a "MENU" with options: "Teacher Processes", "Student Processes", "Add New Student", "Search Student", "Update Student", "Delete Student", "Register To Course", "Enter Exam Results", "List Exam Results", and "Settings". The "Student Processes" section is active, showing the "Add New Student" button. The main content area displays the "Add New Student" form with the following fields:

- Student Number:
- Name:
- Surname:
- Gender:
- Father Name:
- Mother Name:
- Birth Place:
- Birth Date:
- School:

A "Clear" button is located at the bottom right of the form. The status bar at the bottom right shows the time "2:05 PM".

Figure 8.1.10 Add New Student Form

Since user should enter all informations about student, the Add button is placed in the Payment Informations Tab. When a student is added, the student personal informations are added to the "student" table, course informations are added to "crs" table and payment informations are added in "payment_schedule" and calculated the installments and added these are added to the installments table.

Now lets look to the Search Student form in **Figure 8.1.11**.

Supplementary Educational Organization Software :: Department: KADIKÖY UserName: admin - [Search]

Teacher Processes Student Processes Sign Out

MENU

Teacher Processes

Student Processes

Add New Student

Search Student

Update Student

Delete Student

Register To Course

Enter Exam Results

List Exam Results

Settings

All Students Email Unregistered Deleted Other

SNo	name	surname	sex	Address	CityName	postcode	Gsm
20010391	CANER	ÇAKIR	M	address	ISTANBUL	2342	
20123	DENEME	DE	M		ISTANBUL		
12312	OSMAN	B	M	sdfs	ISTANBUL	234213	1212
20010600	MUHAMMED	AKGÜN	M		ISTANBUL		
20020192	OLGAÇ	OCEK	M	address	ISTANBUL	0532	

StudentNo No: Search

Name & Surname Name: Surname: Search

Class

Course Type: COMPUTER

Course: Search

Add Update Delete Details Send Mail Register Payments

3:17 PM

Figure 8.1.11 Search Student Form

In Search Student Form you see a list of all students in a datagrid. Besides filtering by email, unregistered, deleted you can also search by student no, name & surname and by course. Add, update, delete processes is done with the buttons below. All buttons call other forms to perform the necessary actions. Student details can be seen by clicking the Detail button. Unregistered students can be registered with Register button. Payments can be done for any student. And lastly email is sent by clicking the Send Mail button.

Detail form is similar to the Add New Student form but this time the fields are disabled. It is just to list the information and see them in a tabbed view. Tabs are same with in Add New Student Form. But this time course Information and Payment Information is to list the courses which student takes and the payments which student have, in datagrids. Detail Form is shown in **Figure 8.1.12**.

In Personal Informations tab there is an button named "Print". You can print preview the student informations and print the page. By clicking on the button will create crystalreport and list the student informations on it.

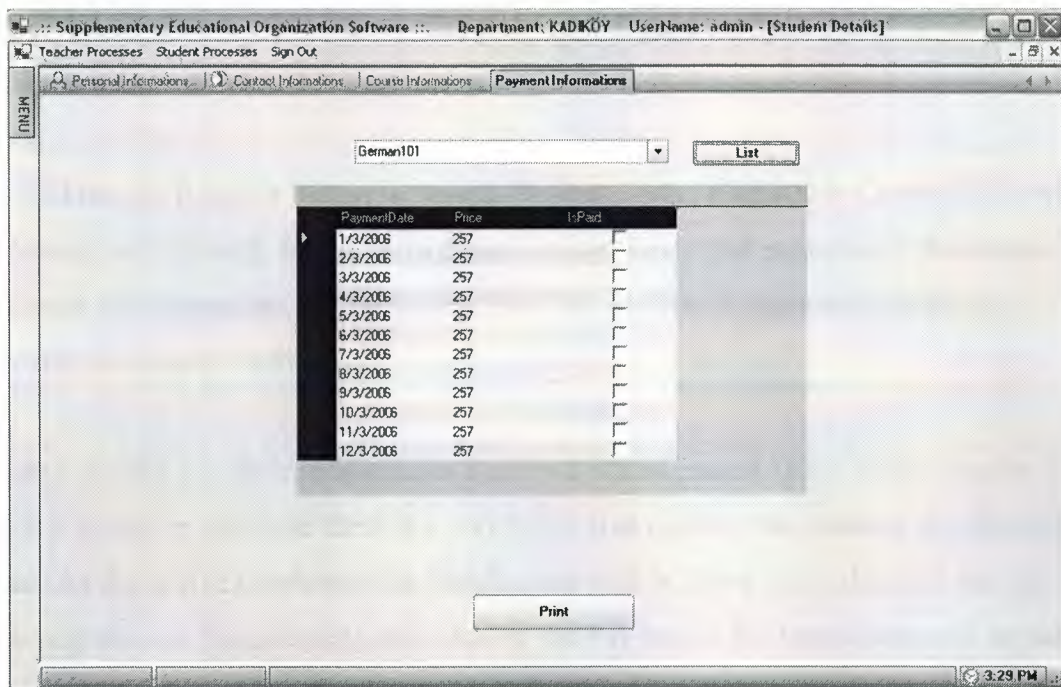


Figure 8.1.12 Student Detail Form Payment Tab

You now see Payment Informations tab in **Figure 8.1.12**. The payment installments are listed in the datagrid according to the course selected which student takes in the combobox. You can print the installment list by clicking the Print button below the datagrid. **Figure 8.1.13** shows what you will see in the payment installments report.

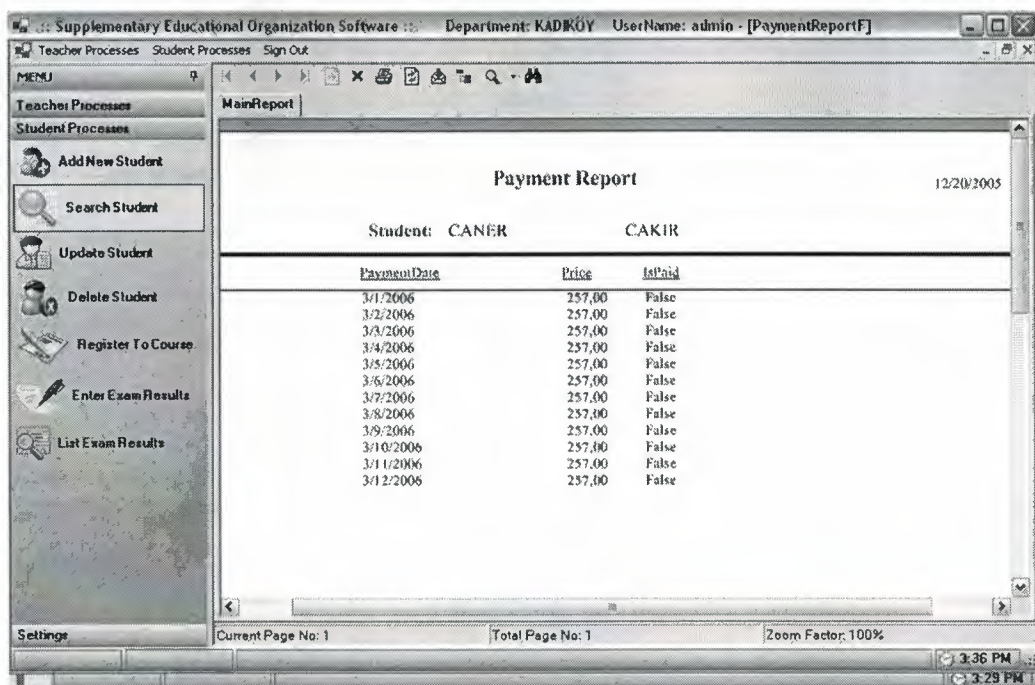


Figure 8.1.13 Payment Installments Report

The crystalreport's toolbar can be used to zoom the page , refresh , print or find a word in the report.

By clicking the Register button in Search Student Form , Register to Course Form will be created and showed. In this form there is a combobox that contains all the courses available for registration for the selected student. Student is registered simply by clicking the Register button.

To let a student pay their installments there is a button named “Pay” on the Search Student Form. In this form there is a combobox that contains the courses that student takes. As the course is selected the installments will be listed in the datagrid and by selecting the row (installment) and clicking the Pay button the installment will be paid. Notice that after payment finish the IsPaid field in datagrid is becomes checked that shows the current installment is paid. (See **Figure 8.1.14**)

InstallmentId	CourseName	PaymentDate	Price	IsPaid
14	German101	1/3/2006	257	<input checked="" type="checkbox"/>
15	German101	2/3/2006	257	<input type="checkbox"/>
16	German101	3/3/2006	257	<input type="checkbox"/>
17	German101	4/3/2006	257	<input type="checkbox"/>
18	German101	5/3/2006	257	<input type="checkbox"/>
19	German101	6/3/2006	257	<input type="checkbox"/>
20	German101	7/3/2006	257	<input type="checkbox"/>
21	German101	8/3/2006	257	<input type="checkbox"/>
22	German101	9/3/2006	257	<input type="checkbox"/>
23	German101	10/3/2006	257	<input type="checkbox"/>
24	German101	11/3/2006	257	<input type="checkbox"/>

Figure 8.1.14 Payment Installments Report

Lets see now the button named “**Enter Exam Results**” on the main menu. This button will create and call a form which lists exams in a datagrid. And when a exam is selected on the datagrid the students who took that exam are listed in a combobox.

(See **Figure 8.1.15**)

ExamId	CourseName	ExamDate	ExamPlace	ExamTime	CourseId
2	VB Programmi	6/18/2006	Room12	6/12/2006	3
3	Geman101	12/26/2005	LAB20	12/30/1899	2

Exam :

Student No:

Student Name:

Exam Result:

Figure 8.1.15 Enter Exam Results Form

As the student selected from the combobox then the name and surname appears in the disabled textbox. Then by giving the exam results to the NumericUpDown control and then pressing the Update button will write the result of the exam to the student.

Last Process for the Student Processes tab is the “List Exam Results”. Lets see what that button will do. As you press this button a form named ListExam Results appears. There are 2 combobox , a datagrid and a button on it. Course Types and courses are listed in the comboboxes. And when user select a course (Actually they are exams) the students who entered that exam are listed in the datagrid with their name surname and exam results. (See **Figure 8.1.16**)

By pressing the button named Print, the crystalreport is shown to the user which lists the current exam results of the students in the report and give choice to print the list by clicking on the print button at the crystal report toolbar.



Figure 8.1.16 List Exam Results Form

In Main Menu only there are only Settings tab that we did not discussed yet. Settings tab is a access limited tab as we discussed in previous pages. Only admin users can access to the Settings tab. (See **Figure 8.1.17**)



Figure 8.1.17 Settings Tab in Menu

In settings tab there are bunchs of settings like “Add New City”, “Add New Department”, “Add Exam” , etc. As it is understable from their names what they do is straight forward.

Forms	Description
Add New City	Adds a new city to the City Table in the database.
Add New Department	Adds a new department to the Department table and creates a default user whose username is “admin” and password is “admin”.
Add New User	Adds new user to the Login Table with username and password.
Add New Course	Adds new course to the Courses Table.
Add New Course Type	Adds new course type to the CourseTypes Table
Add Exam	Adds a new exam to the exam table and sends email to the students who takes that course.
Manage Cities	Updates, Deletes cities
Manage Courses	Updates, Deletes courses
Manage Exams	Updates, Deletes exams (change exam time and place)
Manage Users	Updates, Deletes users
Set Interest Rate	Updates the interest rate for the payment calculations for the current department.

8.2 Student Information System

A web site is designed for the students using ASP.NET. The students can sign up , get username and password. And then they can sign in with their username and password. The aim of the website is to give information and let students to see and change their personal informations, see payments and exam results.

Default.aspx is the startup page for the ASP.NET project. See **Figure 8.2.1** In that Web Form Login , Sign Up , Forgot Password Linkbuttons , two textboxes , one for username other for password and a button named Login placed on it. Student may sign up to the Student Information System by clicking “Sign up” Linkbutton.

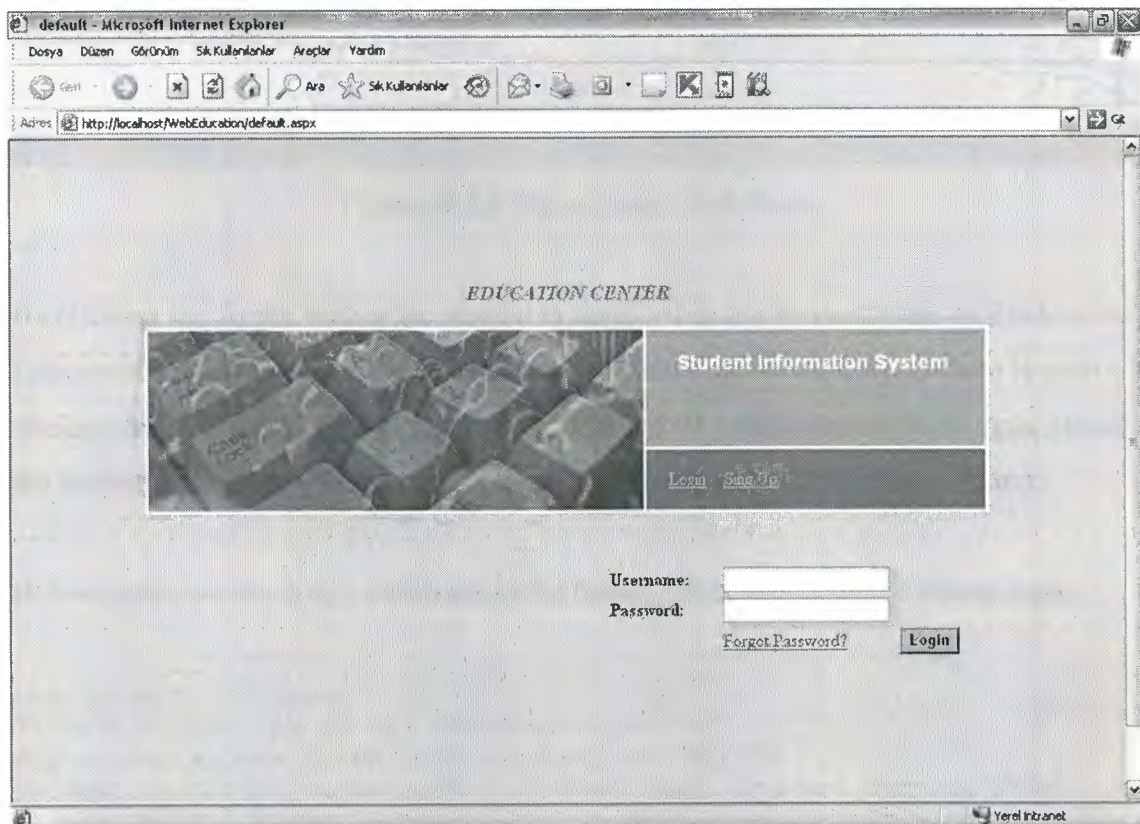


Figure 8.2.1 Default.aspx Web Form

Signup.aspx Web Form contains fields that requires student information to check the student informations that's in the database. See **Figure 8.2.2**.

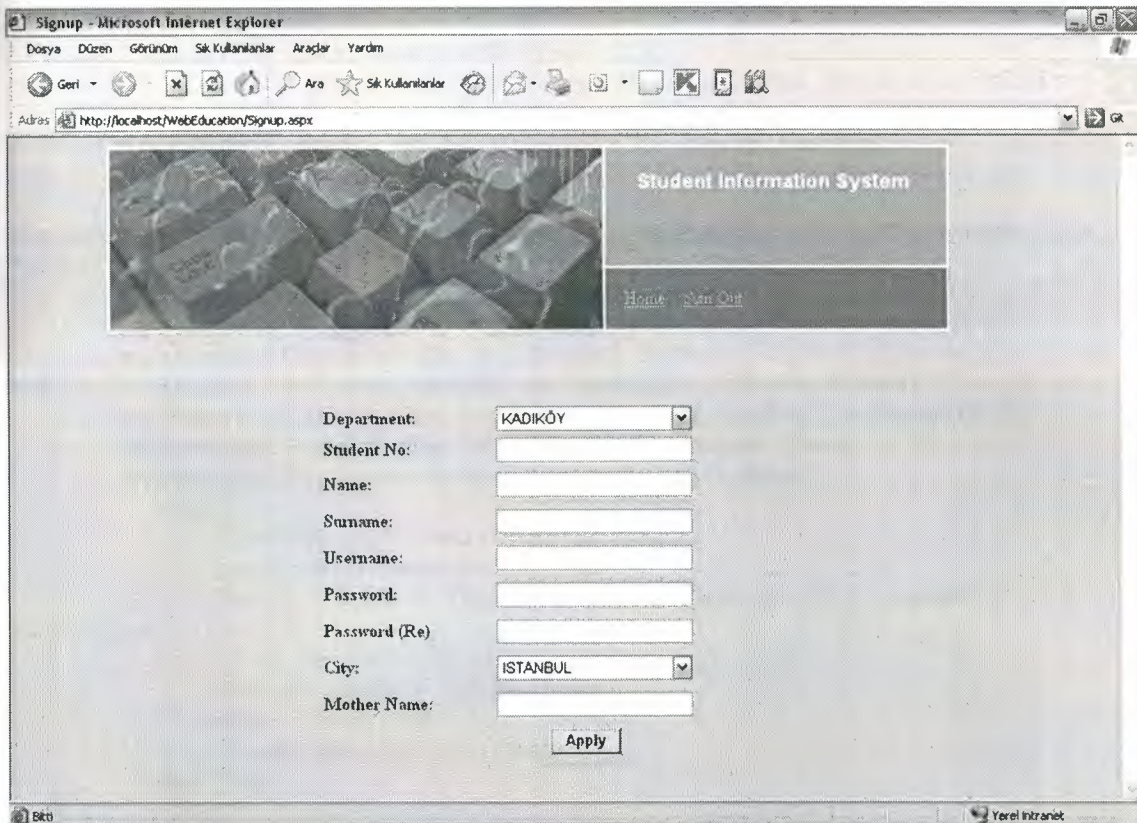


Figure 8.2.2 Signup.aspx Web Form

By clicking the Apply button the student is searched in the Student table by Student No, Department, student name, Student surname, Mother name and City. If there is such student then the username and password is added to the table named WebLogin. Hence the student is registered and can now login into the Student Information System.

Below codes are the codes which are in the button_click procedure of signup.aspx:

```
Dim ds As New DataSet
'Search Student by No and Department Id
Dim myadap As New OleDb.OleDbDataAdapter("SELECT
Student.StudentId,Student.SNo, Student.name, Student.surname FROM
Student WHERE Student.Sno=@Sno and Student.isDeleted=false and
Student.DepartmentId=@DID and Student.name=@name and
student.surname=@surname and student.MotherName=@Mname and
Student.CityId=@City", OleDbConnection1)
myadap.SelectCommand.Parameters.Add("@Sno", Sno.Text)
myadap.SelectCommand.Parameters.Add("@DID", Department.SelectedValue)
myadap.SelectCommand.Parameters.Add("@name", SName.Text)
myadap.SelectCommand.Parameters.Add("@surname", Ssurname.Text)
myadap.SelectCommand.Parameters.Add("@Mname", Mname.Text)
myadap.SelectCommand.Parameters.Add("@City", CityList.SelectedValue)
myadap.Fill(ds)
```



```

'If there is no such Student
If ds.Tables(0).Rows.Count = 0 Then
Result.Text = "No Such Student Found Please Check Your Information"
Else
'If student Found write password to table WebLogin
Dim encrypted As String
encrypted =
FormsAuthentication.HashPasswordForStoringInConfigFile(Password.Text,
"sha1")
Dim mycommand As New OleDb.OleDbCommand
mycommand.Connection = OleDbConnection1
mycommand.CommandText = "INSERT INTO
WebLogin(StudentID,Uname,UPass) VALUES(@Sid,@uname,@pass)"
mycommand.Parameters.Add("@Sid", (ds.Tables(0).Rows(0)(0)))
mycommand.Parameters.Add("@uname", Uname.Text)
mycommand.Parameters.Add("@pass", encrypted)
Try
OleDbConnection1.Open()
mycommand.ExecuteNonQuery()
Result.Text = "Application is successful please login
with your password"
Catch ex As Exception
Result.Text = ex.Message
Finally
OleDbConnection1.Close()
End Try
End If

```

After signing up in the Student Information System, main.aspx Web Form is appear. This page contains a simple menu at the right. There are 3 HyperLink on the menu. Student Information, Payment Information, Exam Results are the HyperLink buttons' Text. Main.aspx Web Form is shown in **Figure 8.2.3**.

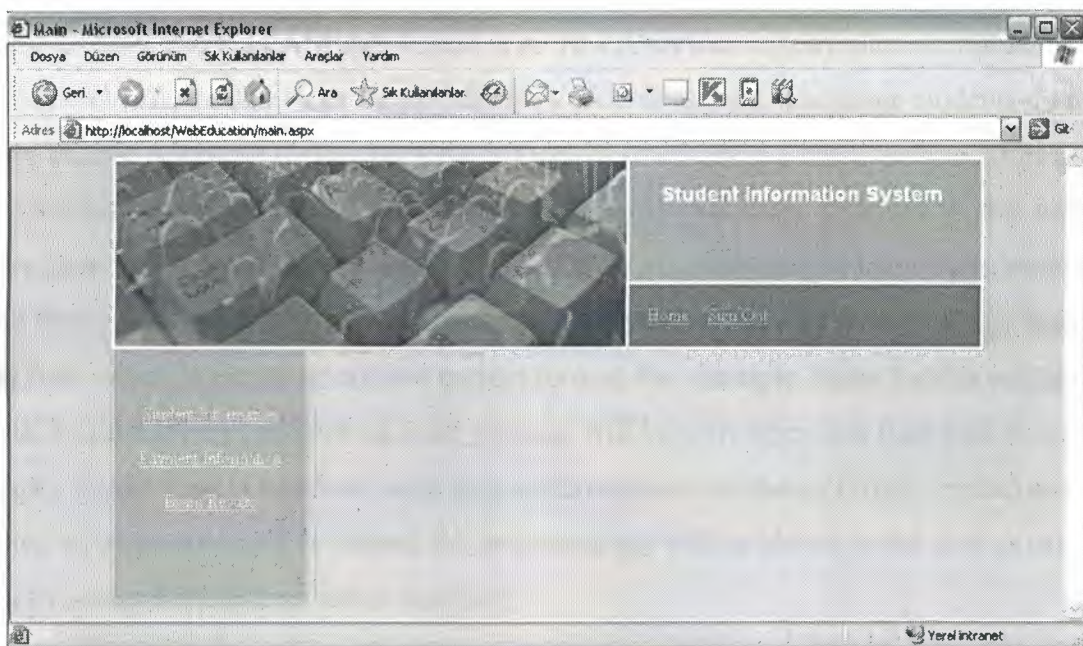


Figure 8.2.3 Main.aspx Web Form

Figure 8.2.4 shows the Student Information page. Personal informations of the student are listed. And student can change and update his/her personal informations. Also password can be changed.

The screenshot shows a web browser window titled 'SinfoPage - Microsoft Internet Explorer'. The address bar shows 'http://localhost:1024/SinfoPage/StudentInformation.aspx'. The page content includes a header with 'Student Information System' and a navigation menu with 'Home' and 'Student'. The main content area displays the following student information:

Student No:	20010391
Student Name:	CANER
Student Surname:	ÇAKIR
Gender:	<input checked="" type="radio"/> M <input type="radio"/> F
Username:	caner_cakir
Password:	
Address:	address
City:	ISTANBUL
Postcode:	
Gsm:	234234
Phone:	242342
Email:	caner_cakir@yahoo.com

There are two buttons: 'Change Password' and 'Update'.

Figure 8.2.4 Student Information Web Form

In windows forms there are ErrorProviders to check some controls content and give user some error message. In ASP.NET there is no ErrorProvider instead, there is validators. In Student Information Page RequiredField Validator is used. Therefore students do not have chance to update his/her personal informations empty and since the validators will show error messages on lost focus from the field which validator is binded to, this will save time for the user. Otherwise student will enter all fields maybe leave some empty and then preses the button named Update. Then he/she will see an error message beside the field which is empty or not in a correct format. For example Name field is validated with RequiredField validator so error message will be seen when this field will be left empty, Email field is validated with RegularExpression validator (Email Format) and when an incorrect email is entered the error message will be shown to the user as user clicks somewhere else or leaves that field.

Below code is the code block for the Update Button click event.

```
'Updating Student Info
Dim mycommand As New OleDb.OleDbCommand("Update Student set
name=@name ,surname=@sname , sex=@gender, address=@addres,
cityId=@cID, postcode=@pc, gsm=@gsm, phone=@phone, email=@email where
studentId=@SID", OleDbConnection1)
mycommand.Parameters.Add("@name", Sname.Text)
mycommand.Parameters.Add("@sname", Ssurname.Text)
If M.Checked = True Then
    mycommand.Parameters.Add("@gender", M.Text)
Else
    mycommand.Parameters.Add("@gender", F.Text)
End If
mycommand.Parameters.Add("@address", Address.Text)
mycommand.Parameters.Add("@cID", City.SelectedValue)
mycommand.Parameters.Add("@pc", Postcode.Text)
mycommand.Parameters.Add("@gsm", Gsm.Text)
mycommand.Parameters.Add("@phone", Phone.Text)
mycommand.Parameters.Add("@email", Email.Text)
mycommand.Parameters.Add("SID", Session("SID"))
Try
    OleDbConnection1.Open()
    mycommand.ExecuteNonQuery()

Catch ex As Exception
    result.Text = ex.Message
Finally
    OleDbConnection1.Close()
End Try
```

Figure 8.2.5 Shows the Payment Information page of the Student Information System.

Student Information System

Please Select Course

German101

Show Payments

PaymentDate	Price	IsPaid
03.01.2006 00:00:00	257	True
03.02.2006 00:00:00	257	False
03.03.2006 00:00:00	257	False
03.04.2006 00:00:00	257	False
03.05.2006 00:00:00	257	False
03.06.2006 00:00:00	257	False
03.07.2006 00:00:00	257	False
03.08.2006 00:00:00	257	False
03.09.2006 00:00:00	257	False
03.10.2006 00:00:00	257	False
03.11.2006 00:00:00	257	False
03.12.2006 00:00:00	257	False

Figure 8.2.5 Payment Information Web Form

In this Web Form since the students can take more than one course, first course should be selected. The courses that student takes are listed in the combobox on page load. To see the payment installment list, the course is selected from the course and the button named “Show Payment” is pressed. The Figure 8.2.5 is the result after pressing button. Below code is written in the button click event to list the payment installments.

```
Dim myadapter As New OleDb.OleDbDataAdapter("SELECT Student.Sno As
Student_No,Student.name As
Name,Student.Surname,Exam.ExamDate,ExamResults.Result FROM Student
INNER JOIN (Course INNER JOIN (Exam INNER JOIN ExamResults ON
Exam.ExamId = ExamResults.ExamId) ON Course.CourseId = Exam.CourseId)
ON Student.StudentId = ExamResults.StudentId where
Student.StudentId=@SID and Course.CourseId=@CID;", OleDbConnection1)
myadapter.SelectCommand.Parameters.Add("@SID", Session("sid"))
myadapter.SelectCommand.Parameters.Add("@CID",
Exams.SelectedValue)
Dim ds As New DataSet
myadapter.Fill(ds)
DataGrid1.DataSource = ds.Tables(0)
DataGrid1.DataBind()
```

Figure 8.2.6 shows the Exam Results Form. Simply the courses that student takes are listed in the combobox and after the button named “Show Exam Result” is pressed the student’s exam result is listed for that course in datagrid.

Student_No	Name	Surname	ExamDate	Result
20010391	CANER	ÇAKIR	26.12.2005 00:00:00	86

Figure 8.2.6 List Exam Results Web Form

Lets see what will happen if a student forgets his/her password. There is a Linkbutton on the default.aspx named “Forgot password?” (see **Figure 8.2.1**) This Linkbutton will redirect the student to the Forgot password Web Form.

Before disscussing the Forgot password Web Form, it is needed to talk a little about **Encryption**. The passwords in the database is kept in “**sha1**” encryption technique which is not possible to rollback. So it is not possible to ask student his/her information and give his/her password. The only way to give student a new password which means reseting the previous one and creating a new password with a userdefined procedure. And new password is then sent to the student’s email. In **Figure 8.2.7** Forgot Password Web Form is shown.

Figure 8.2.7 Forgot Password Web Form

The userdefined procedure to create a new password is below:

```
Dim random1 As New Random
Dim pass As String
Dim myarray() As String = {"A", "D", "F", "Z", "T", "1", "5", "7",
    "K", "4", "L", "Y", "U", "M", "4", "8"}
Dim number As Integer
For i As Integer = 1 To 5
    number = random1.Next(1, 16)
    pass &= myarray(number)
Next
```

This procedure has an array of string named myarray and it contains some alphabetical characters and numeric characters. A Random variable declared to generate random numbers between 1 to 16. For loop is from 1 to 16 because there are 16 characters in the array named myarray. So a randomly selected number between 1 and 16 means randomly selected character from the array myarray. Then any character selected are added to the variable pass which has string datatype.

Last thing to do with the password is to encrypt it again before writing it to the table. Below code will show how it is encrypted .

```
Dim encrypted As String
encrypted =
FormsAuthentication.HashPasswordForStoringInConfigFile(pass, "sha1")
```

“sha1” encryption technique is used to encrypt the password. If specialist forgets that little point , this user will get the password to his/her email and will not be able to login because as he/she writes the password, it will encrypted and will be compared with the one in the database and the bad news is the password in the database was not encrypted before write process.

Now it is safe to write the password to the table. After writing the password to the weblogin table now it is the time to make student know his/her new password. Below code shows how to send email.

```
System.Web.Mail.SmtpMail.Send("Education Center",
ds.Tables(0).Rows(0)("Email"), "Your New Password", pass)
```

CONCLUSION

The aim of my project is to create an alternative for organization software. It proves that an organization software can work without an local server. Since internet connection is required, the hosting server is used only. Therefore it saves from buying and maintaining the server machines that cost much and make problems.

It is not told that there is no problem with the web service projects. The situation of same time processes and conflicts may occur in real practice. But these problems can be solved by using other database technologies. Sql server can overcome these problems. Another problem is that, since application is communicating with web service and perform actions on the web the operation times are slower than an local one. Higher internet connections can overcome these problem.

Hence I believe that the technology I used will be very popular in big companies and the problems I face to face while doing that project will not be a big deal in near future.

REFERENCES

Balena, Francesco,(2004) Programming Microsoft Visual Basic .NET version 2003:Microsoft

VB.NET Informations from

<http://www.dotnetjunkies.com>

ASP.NET Informations from

<http://www.asp.net>

APPENDIX A: PROGRAM CODES

Service1.asmx

```
Imports System.Web.Services
Imports System.Web.Security
Imports System.Data.OleDb
<System.Web.Services.WebService(Namespace:="http://tempuri.org/Webservisim/
Service1")> _
Public Class Service1
    Inherits System.Web.Services.WebService

    #Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code after the InitializeComponent()
call

    End Sub

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Web Services Designer
    'It can be modified using the Web Services Designer.
    'Do not modify it using the code editor.
    Friend WithEvents OleDbConnection1 As System.Data.OleDb.OleDbConnection
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Me.OleDbConnection1 = New System.Data.OleDb.OleDbConnection
        ,
        'OleDbConnection1
        ,
        Me.OleDbConnection1.ConnectionString = "Jet OLEDB:Global Partial
Bulk Ops=2;Jet OLEDB:Registry Path=;Jet OLEDB:Database L" & _
```

```

        "locking Mode=1;Jet OLEDB:Database Password=;Data
Source="C:\Inetpub\wwwroot\Webse" & _
        "rvisim\Dersane.mdb";Password=;Jet OLEDB:Engine Type=5;Jet
OLEDB:Global Bulk Tran" & _
        "sactions=1;Provider="Microsoft.Jet.OLEDB.4.0";Jet OLEDB:System
database=;Jet OLE" & _
        "DB:SFP=False;Extended Properties=;Mode=Share Deny None;Jet
OLEDB:New Database Pa" & _
        "ssword=;Jet OLEDB:Create System Database=False;Jet OLEDB:Don't
Copy Locale on Co" & _
        "mpact=False;Jet OLEDB:Compact Without Replica Repair=False;User
ID=Admin;Jet OLE" & _
        "DB:Encrypt Database=False"

```

End Sub

Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)

'CODEGEN: This procedure is required by the Web Services Designer

'Do not modify it using the code editor.

If disposing Then

 If Not (components Is Nothing) Then

 components.Dispose()

 End If

End If

MyBase.Dispose(disposing)

End Sub

#End Region

'http://localhost/Webservisim/Service1.asmx

'http://tempuri.org/Webservisim/Service1

' WEB SERVICE EXAMPLE

' The HelloWorld() example service returns the string Hello World.

' To build, uncomment the following lines then save and build the project.

' To test this web service, ensure that the .asmx file is the start page

' and press F5.

'

'<WebMethod()> _

'Public Function HelloWorld() As String

' Return "Hello World"

'End Function

'List All Students to the DataGrid in Search Form

<WebMethod()> _

Public Function ListStudents(ByVal DID As Integer) As DataSet

```
    Dim myadap As New OleDbDataAdapter("SELECT Student.SNo,
Student.name, Student.surname, Student.sex, Student.Address, City.CityName,
Student.postcode, Student.Gsm, Student.OsymNo, Student.phone,
Student.Email, Advisor.AdvisorName, Student.StudentId FROM Advisor INNER
JOIN (City INNER JOIN Student ON City.CityId = Student.CityId) ON
Advisor.AdvisorId = Student.AdvisorId where Student.isDeleted=false and
Student.DepartmentId=@DID;", OleDbConnection1)
```

```
    myadap.SelectCommand.Parameters.Add("@DID", DID)
```

```
    Dim ds As New DataSet
```

```
    myadap.Fill(ds)
```

```
    Return (ds)
```

End Function

'Search a Student with his/her Student No

<WebMethod()> _

Public Function SearchStudent(ByVal Sno As Integer, ByVal DID As Integer) As DataSet

```
    Dim myadap As New OleDbDataAdapter("SELECT Student.SNo,
Student.name, Student.surname, Student.sex, Student.Address, City.CityName,
Student.postcode, Student.Gsm, Student.OsymNo, Student.phone,
Student.Email, Advisor.AdvisorName, Student.StudentId FROM Advisor INNER
JOIN (City INNER JOIN Student ON City.CityId = Student.CityId) ON
Advisor.AdvisorId = Student.AdvisorId WHERE Student.Sno=@Sno and
Student.isDeleted=false and Student.DepartmentId=@DID", OleDbConnection1)
```

```
    myadap.SelectCommand.Parameters.Add("@Sno", Sno)
```

```
    myadap.SelectCommand.Parameters.Add("@DID", DID)
```

```
    Dim ds As New DataSet
```

```
    myadap.Fill(ds)
```

```
    Return (ds)
```

End Function

'Search a Student with his/her Name and Surname

<WebMethod()> _

Public Function SearchByName(ByVal Name As String, ByVal Surname As String, ByVal DID As Integer) As DataSet

```

Dim myadap As New OleDbDataAdapter("SELECT Student.SNo,
Student.name, Student.surname, Student.sex, Student.Address, City.CityName,
Student.postcode, Student.Gsm, Student.OsymNo, Student.phone,
Student.Email, Advisor.AdvisorName, Student.StudentId FROM Advisor INNER
JOIN (City INNER JOIN Student ON City.CityId = Student.CityId) ON
Advisor.AdvisorId = Student.AdvisorId WHERE Student.name=@Name and
Student.surname=@Surname and Student.isDeleted=false and
Student.DepartmentId=@DID", OleDbConnection1)

myadap.SelectCommand.Parameters.Add("@Name", Name)
myadap.SelectCommand.Parameters.Add("@Surname", Surname)
myadap.SelectCommand.Parameters.Add("@DID", DID)

Dim ds As New DataSet
myadap.Fill(ds)
Return (ds)

End Function

```

'Function to Add New Student to Student Table only

```
<WebMethod()> _
```

```

Public Function Addstudent(ByVal name As String, ByVal surname As
String, ByVal Sno As Integer, ByVal school As String, ByVal sex As String,
ByVal address As String, ByVal cityId As Integer, ByVal postcode As String,
ByVal gsm As String, ByVal fname As String, ByVal mname As String, ByVal
bplace As String, ByVal bdate As Date, ByVal OSYM As String, ByVal notes As
String, ByVal email As String, ByVal phone As String, ByVal advisor As
Integer, ByVal DID As Integer) As String

Dim komutum As New OleDb.OleDbCommand
komutum.Connection = OleDbConnection1
komutum.CommandText = "Insert into
Student (DepartmentId, AdvisorId, SNo, name, surname, school, sex, Address, CityId, p
ostcode, Gsm, FatherName, MotherName, BirthPlace, BirthDate, OsymNo, Notes, Email, p
hone)
values (@DID, @AID, @Sno, @name, @surname, @school, @sex, @address, @city, @postcode,
@Gsm, @fname, @mname, @bplace, @bdate, @OSYM, @notes, @email, @phone) "

komutum.Parameters.Add("@DID", DID)
komutum.Parameters.Add("@AID", advisor)
komutum.Parameters.Add("@Sno", Sno)
komutum.Parameters.Add("@name", name)
komutum.Parameters.Add("@surname", surname)
komutum.Parameters.Add("@school", school)
komutum.Parameters.Add("@sex", sex)
komutum.Parameters.Add("@address", address)

```

```

komutum.Parameters.Add("@city", cityId)
komutum.Parameters.Add("@postcode", postcode)
komutum.Parameters.Add("@Gsm", gsm)
komutum.Parameters.Add("@fname", fname)
komutum.Parameters.Add("@mname", mname)
komutum.Parameters.Add("@bplace", bplace)
komutum.Parameters.Add("@bdate", bdate)
komutum.Parameters.Add("@OSYM", OSYM)
komutum.Parameters.Add("@notes", notes)
komutum.Parameters.Add("@email", email)
komutum.Parameters.Add("@phone", phone)
Dim myadap As New OleDbDataAdapter("Select
StudentId,Sno,name,surname from Student WHERE Sno=@no and name=@name and
surname=@surname", OleDbConnection1)
myadap.SelectCommand.Parameters.Add("@no", Sno)
myadap.SelectCommand.Parameters.Add("@name", name)
myadap.SelectCommand.Parameters.Add("@surname", surname)
Dim ds As New DataSet
Try
    OleDbConnection1.Open()
    komutum.ExecuteNonQuery()
    myadap.Fill(ds)
    Dim s As String
    s = ds.Tables(0).Rows(0)("StudentId")
    Return s
Catch ex As Exception
Finally
    OleDbConnection1.Close()
End Try
End Function

```

'Function to Record the Payment Informations

```

<WebMethod()> _
Public Function RecordPayment(ByVal SId As Integer, ByVal total As
Single, ByVal Advanced As Single, ByVal Remaining As Single, ByVal
Installments As Integer, ByVal CourseId As Integer) As String
    Dim mycomm As New OleDbCommand
    mycomm.Connection = OleDbConnection1

```



```

        mycomm.CommandText = "INSERT INTO
PaySchedule(StudentId,Total,InAdvanced,Remaining,NoOfInstallment,CourseId)
Values(@SID,@Total,@Advanced,@Remaining,@Installment,@CId)  "
        mycomm.Parameters.Add("@SID", SID)
        mycomm.Parameters.Add("@Total", total)
        mycomm.Parameters.Add("@Advanced", Advanced)
        mycomm.Parameters.Add("@Remaining", Remaining)
        mycomm.Parameters.Add("@Installment", Installments)
        mycomm.Parameters.Add("@CId", CourseId)
    Try
        OleDbConnection1.Open()
        mycomm.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

'Function to Record the Payment Installments

```

<WebMethod()> _
Public Function RecordInstallment(ByVal SId As Integer, ByVal total As
Single, ByVal Advanced As Single, ByVal Remaining As Single, ByVal
Installments As Integer, ByVal Pdate As Date, ByVal Price As Single) As
String
    Dim mycomm As New OleDbCommand
    Dim mycomm2 As New OleDbCommand
    mycomm.Connection = OleDbConnection1
    mycomm2.Connection = OleDbConnection1
    mycomm.CommandText = "INSERT INTO
Installment(PaymentId,PaymentDate,Price)
Values(@PaymentId,@PaymentDate,@Price)  "
    mycomm2.CommandText = "Select PaymentId from PaySchedule where
StudentId=@SID and total=@Total and Remaining =@Remain and
NoOfInstallment=@Installments"
    mycomm2.Parameters.Add("@SID", SId)
    mycomm2.Parameters.Add("@Total", total)
    mycomm2.Parameters.Add("@Remain", Remaining)
    mycomm2.Parameters.Add("@Installments", Installments)
    Try
        OleDbConnection1.Open()

```

```

        mycomm.Parameters.Add("@PaymentId", mycomm2.ExecuteScalar)
        mycomm.Parameters.Add("@PaymentDate", Pdate)
        mycomm.Parameters.Add("@Price", Price)
        mycomm.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

'Function to Add New Advisor

```
<WebMethod()> _
```

```

Public Function AddAdvisor(ByVal name As String, ByVal surname As
String, ByVal phone As String, ByVal address As String, ByVal DID As
Integer) As String

```

```

    Dim mycommand As New OleDb.OleDbCommand
    mycommand.Connection = OleDbConnection1
    mycommand.CommandText = "Insert into

```

```

Advisor (AdvisorName, AdvisorSurname, Phone, Address, DepartmentId)
Values (@N, @S, @P, @A, @D) "

```

```

    mycommand.Parameters.Add("@N", name)
    mycommand.Parameters.Add("@S", surname)
    mycommand.Parameters.Add("@P", phone)
    mycommand.Parameters.Add("@A", address)
    mycommand.Parameters.Add("@D", DID)

```

```
Try
```

```

    OleDbConnection1.Open()
    mycommand.ExecuteNonQuery()

```

```
Catch ex As Exception
```

```
Return ex.Message
```

```
Finally
```

```
OleDbConnection1.Close()
```

```
End Try
```

```
End Function
```

'Fuction to list all advisor teachers

```
<WebMethod()> _
```

```
Public Function ListAdvisors(ByVal DID As Integer) As DataSet
```

```

        Dim myadapter As New OleDb.OleDbDataAdapter("Select
AdvisorId,AdvisorName+' '+AdvisorSurname As NameSurname,phone,Address from
Advisor where DepartmentId=@DID and isDeleted=false", OleDbConnection1)
        myadapter.SelectCommand.Parameters.Add("@DID", DID)
        Dim ds As New DataSet
        myadapter.Fill(ds)
        Return (ds)
    End Function

```

'Function to list all Cities

```

    <WebMethod()> _
    Public Function Listcities() As DataSet
        Dim myadapter As New OleDb.OleDbDataAdapter("Select CityId,CityName
from City", OleDbConnection1)
        Dim ds As New DataSet
        myadapter.Fill(ds)
        Return (ds)
    End Function

```

'Function to Update City

```

    <WebMethod()> _
    Public Function Updatecity(ByVal CID As Integer, ByVal name As String)
As String
        Dim mycommand As New OleDb.OleDbCommand("UPDATE City SET
CityName=@name WHERE CityId=@CID", OleDbConnection1)
        mycommand.Parameters.Add("@name", name)
        mycommand.Parameters.Add("@CID", CID)
        Try
            OleDbConnection1.Open()
            mycommand.ExecuteNonQuery()
        Catch ex As Exception
            Return ex.Message
        Finally
            OleDbConnection1.Close()
        End Try
    End Function

```

'Function to Delete City

```

    <WebMethod()> _
    Public Function Delcity(ByVal CID As Integer) As String

```



```

        Dim mycommand As New OleDb.OleDbCommand("DELETE FROM City WHERE
CityId=@CID", OleDbConnection1)
        mycommand.Parameters.Add("@CID", CID)
        Try
            OleDbConnection1.Open()
            mycommand.ExecuteNonQuery()
        Catch ex As Exception
            Return ex.Message
        Finally
            OleDbConnection1.Close()
        End Try
    End Function

```

'Function to Add New City

```

<WebMethod()> _
Public Function AddCity(ByVal city As String) As String
    Dim komutum As New OleDb.OleDbCommand
    komutum.Connection = OleDbConnection1
    komutum.CommandText = "Insert into City(CityName) Values(@city)"
    komutum.Parameters.Add("@city", city)
    Try
        OleDbConnection1.Open()
        komutum.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

'List Departments

```

<WebMethod()> _
Public Function ListDepartments() As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("Select
DepartmentId,DepartmentName from Department", OleDbConnection1)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function

```

'Check Username Password And Department in LOGIN

<WebMethod()> _

```
Public Function CheckIt(ByVal Uname As String, ByVal Password As
String, ByVal DID As Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT Login.UserId,
Login.UserName, Login.UserPassword, Login.IsAdmin FROM Department INNER
JOIN Login ON Department.DepartmentId = Login.DepartmentId where
Login.DepartmentId=@DID and Login.UserName=@Uname And
Login.UserPassword=@Password", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    myadapter.SelectCommand.Parameters.Add("@Uname", Uname)
    myadapter.SelectCommand.Parameters.Add("@Password", Password)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return ds
End Function
```

'ADD Department

<WebMethod()> _

```
Public Function AddDepartment(ByVal name As String, ByVal address
As String, ByVal phone As String, ByVal fax As String, ByVal Irate As
Integer) As String
    Dim mycommand As New OleDb.OleDbCommand
    Dim mycommand2 As New OleDb.OleDbCommand
    Dim mycommand3 As New OleDb.OleDbCommand
    mycommand.Connection = OleDbConnection1
    mycommand2.Connection = OleDbConnection1
    mycommand3.Connection = OleDbConnection1
    mycommand.CommandText = "Insert into
Department (DepartmentName,Address,Phone,Fax,InterestRate)
Values (@name,@address,@phone,@fax,@Irate) "
    mycommand.Parameters.Add("@name", name)
    mycommand.Parameters.Add("@address", address)
    mycommand.Parameters.Add("@phone", phone)
    mycommand.Parameters.Add("@fax", fax)
    mycommand.Parameters.Add("@Irate", Irate)
    mycommand2.CommandText = "Select DepartmentId from Department where
DepartmentName=@name and Address=@address"
    mycommand2.Parameters.Add("@name", name)
    mycommand2.Parameters.Add("@address", address)
```

```

Dim encodedpass As String
encodedpass =
FormsAuthentication.HashPasswordForStoringInConfigFile("admin", "sha1")
Try
    OleDbConnection1.Open()
    mycommand.ExecuteNonQuery()
    mycommand3.CommandText = "Insert Into
Login(Username,UserPassword,DepartmentId,Isadmin)
Values(@Uname,@Upass,@DeptID,@Isadmin) "
    mycommand3.Parameters.Add("Uname", "admin")
    mycommand3.Parameters.Add("Upass", encodedpass)
    mycommand3.Parameters.Add("@DeptId", mycommand2.ExecuteScalar)
    mycommand3.Parameters.Add("Isadmin", True)
    mycommand3.ExecuteNonQuery()
Catch ex As Exception
    Return (ex.Message())
Finally
    OleDbConnection1.Close()
End Try
End Function

```

'Create the MainPage name as department name and username

```

<WebMethod()> _
Public Function MainPageName(ByVal DID As Integer, ByVal UID As
Integer) As String
    Dim resultname As String
    Dim myadapter As New OleDb.OleDbDataAdapter("Select DepartmentName
from Department where DepartmentId=@DID", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    resultname = ".:: Supplementary Educational Organization Software
::: Department: " & ds.Tables(0).Rows(0)("DepartmentName")
    Dim myadapter2 As New OleDb.OleDbDataAdapter("Select UserName from
Login where UserId=@UID", OleDbConnection1)
    myadapter2.SelectCommand.Parameters.Add("@UID", UID)
    Dim ds2 As New DataSet
    myadapter2.Fill(ds2)
    resultname += " UserName: " & ds2.Tables(0).Rows(0)("UserName")
    Return resultname
End Function

```



```

<WebMethod()> _
    Public Function CheckUser(ByVal uname As String, ByVal pass As String,
ByVal DID As Integer, ByVal Isadmin As Boolean) As Boolean
        'Check if There is Same User in The department
        Dim myadapter As New OleDb.OleDbDataAdapter("Select UserId from
Login where UserName=@uname and UserPassword=@pass and DepartmentId=@DID",
OleDbConnection1)

        myadapter.SelectCommand.Parameters.Add("@uname", uname)
        myadapter.SelectCommand.Parameters.Add("@pass", pass)
        myadapter.SelectCommand.Parameters.Add("@DID", DID)
        Dim ds As New DataSet
        myadapter.Fill(ds)
        'If there is same user in department return = false else return =
True
        If ds.Tables(0).Rows.Count <> 0 Then
            Return False
        Else
            Return True
        End If
    End Function

    'Add New User
    <WebMethod()> _
        Public Function AddUser(ByVal uname As String, ByVal pass As String,
ByVal DID As Integer, ByVal Isadmin As Boolean) As String
            Dim mycommand As New OleDb.OleDbCommand
            mycommand.Connection = OleDbConnection1
            mycommand.CommandText = "Insert into
Login(UserName,UserPassword,DepartmentId,IsAdmin)
Values(@Uname,@pass,@DID,@Admin) "
            mycommand.Parameters.Add("@Uname", uname)
            mycommand.Parameters.Add("@pass", pass)
            mycommand.Parameters.Add("@DID", DID)
            mycommand.Parameters.Add("@Admin", Isadmin)
            Try
                OleDbConnection1.Open()
                mycommand.ExecuteNonQuery()
            Catch ex As Exception
                Return ex.Message
            Finally

```

```

        OleDbConnection1.Close()
    End Try
End Function

```

'Add New Teacher function

```

<WebMethod()> _
Public Function AddTeacher(ByVal name As String, ByVal surname As
String, ByVal address As String, ByVal gsm As String, ByVal phone As
String, ByVal email As String, ByVal DID As Integer, ByVal Bdate As Date)
As String
    Dim mycommand As New OleDb.OleDbCommand
    mycommand.Connection = OleDbConnection1
    mycommand.CommandText = "Insert Into
Teacher (Tname, Tsurname, Address, GSM, Phone, Email, DepartmentId, BirthDate)
Values (@name, @surname, @address, @gsm, @phone, @email, @DID, @bdate) "
    mycommand.Parameters.Add("@name", name)
    mycommand.Parameters.Add("@surname", surname)
    mycommand.Parameters.Add("@Address", address)
    mycommand.Parameters.Add("@gsm", gsm)
    mycommand.Parameters.Add("@phone", phone)
    mycommand.Parameters.Add("@email", email)
    mycommand.Parameters.Add("@DID", DID)
    mycommand.Parameters.Add("@bdate", Bdate)
    Try
        OleDbConnection1.Open()
        mycommand.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

'List All CourseTypes

```

<WebMethod()> _
Public Function ListCourseType() As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("Select
CourseTypeName, CourseTypeId from CourseTypes order by coursetypeName",
OleDbConnection1)
    Dim ds As New DataSet
    myadapter.Fill(ds)

```

```
Return ds
End Function
```

' List All Courses

```
<WebMethod()> _
```

```
Public Function ListCourses(ByVal Id As Integer, ByVal DID As Integer)
As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("Select
CourseId,CourseName from Course where CourseTypeId=@ID", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@ID", Id)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return ds
End Function
```

' List All Courses with all Fields for Manage Courses Form

```
<WebMethod()> _
```

```
Public Function ListCourses_details(ByVal Id As Integer, ByVal DID As
Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("Select
CourseId,CourseName,CourseCode,CourseContent,StartDate,EndDate,IsWeekend,Qu
ota,Price from Course where CourseTypeId=@ID and DepartmentId=@DID",
OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@ID", Id)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return ds
End Function
```

' Update Course With its Id

```
<WebMethod()> _
```

```
Public Function UpdateCourse(ByVal CTYP As Integer, ByVal name As
String, ByVal code As String, ByVal content As String, ByVal Sdate As Date,
ByVal Edate As Date, ByVal IsWeek As Boolean, ByVal quota As Integer, ByVal
price As Single, ByVal CId As Integer, ByVal DID As Integer) As String
    Dim mycommand As New OleDb.OleDbCommand
    mycommand.Connection = OleDbConnection1
```



```

        mycommand.CommandText = "UPDATE Course Set
CourseTypeId=@CTYP,CourseName=@name, CourseCode=@Code,
CourseContent=@Content, StartDate=@Sdate, EndDate=@Edate,
IsWeekend=@IsWeek, Quota=@quota, Price=@price where CourseId=@CID and
DepartmentId=@DID"

```

```

        mycommand.Parameters.Add("@CTYP", CTYP)
        mycommand.Parameters.Add("@name", name)
        mycommand.Parameters.Add("@Code", code)
        mycommand.Parameters.Add("@Content", content)
        mycommand.Parameters.Add("@Sdate", Sdate)
        mycommand.Parameters.Add("@Edate", Edate)
        mycommand.Parameters.Add("@IsWeek", IsWeek)
        mycommand.Parameters.Add("@quota", quota)
        mycommand.Parameters.Add("@price", price)
        mycommand.Parameters.Add("@CID", CId)
        mycommand.Parameters.Add("@DID", DID)

```

```

    Try

```

```

        OleDbConnection1.Open()

```

```

        mycommand.ExecuteNonQuery()

```

```

    Catch ex As Exception

```

```

        Return ex.Message

```

```

    Finally

```

```

        OleDbConnection1.Close()

```

```

    End Try

```

```

End Function

```

' Update Course Type With its Id

```

<WebMethod()> _

```

```

Public Function UpdateCourseType(ByVal CTYP As Integer, ByVal name As
String) As String

```

```

    Dim mycommand As New OleDb.OleDbCommand

```

```

    mycommand.Connection = OleDbConnection1

```

```

    mycommand.CommandText = "UPDATE CourseTypes Set
CourseTypeName=@name where CourseTypeId=@CTYP"

```

```

    mycommand.Parameters.Add("@name", name)

```

```

    mycommand.Parameters.Add("@CTYP", CTYP)

```

```

    Try

```

```

        OleDbConnection1.Open()

```

```

        mycommand.ExecuteNonQuery()

```

```

    Catch ex As Exception

```

```

        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

' Search Students who take specific class

```

<WebMethod()> _
Public Function SearchCourseStudents(ByVal CId As Integer, ByVal DID As
Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter(" SELECT Student.SNo,
Student.name, Student.surname, Student.sex, Student.Address, City.CityName,
Student.postcode, Student.Gsm, Student.OsymNo, Student.phone,
Student.Email, Advisor.AdvisorName, Student.StudentId FROM Course INNER JOIN
((Advisor INNER JOIN (City INNER JOIN Student ON City.CityId =
Student.CityId) ON Advisor.AdvisorId = Student.AdvisorId) INNER JOIN CSR ON
Student.StudentId = CSR.StudentId) ON Course.CourseId = CSR.CourseId WHERE
Course.CourseId=@CID AND Course.DepartmentId=@DID and
Student.isDeleted=false;", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("CID", CId)
    myadapter.SelectCommand.Parameters.Add("DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return ds
End Function

```

' List All Students Who Has Email

```

<WebMethod()> _
Public Function ListStudentEmail(ByVal DID As Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT Student.SNo,
Student.name, Student.surname, Student.sex, Student.Address, City.CityName,
Student.postcode, Student.Gsm, Student.OsymNo, Student.phone,
Student.Email, Advisor.AdvisorName, Student.StudentId FROM Advisor INNER
JOIN (City INNER JOIN Student ON City.CityId = Student.CityId) ON
Advisor.AdvisorId = Student.AdvisorId where Student.Email<>' and
Student.isDeleted=false and Student.DepartmentId=@DID", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return ds
End Function

```

'Delete Student Function (Actually Update Statements changing the isDelete Field Norecord is deleted physically)

<WebMethod()> _

Public Function DelStudent(ByVal SID As Integer, ByVal DelDate As DateTime, ByVal DComment As String) As String

Dim mycommand As New OleDb.OleDbCommand

mycommand.Connection = OleDbConnection1

mycommand.CommandText = "UPDATE Student set isDeleted=true ,
DeletionDate=@DelDate, DeletionComment=@DComment where
Student.StudentId=@SID"

mycommand.Parameters.Add("@DelDate", DelDate)

mycommand.Parameters.Add("@DComment", DComment)

mycommand.Parameters.Add("@SID", SID)

Try

OleDbConnection1.Open()

mycommand.ExecuteNonQuery()

Catch ex As Exception

Return ex.Message

Finally

OleDbConnection1.Close()

End Try

End Function

'Function to Search Student for Details Form

<WebMethod()> _

Public Function ShowStudent_Details(ByVal SID As Integer) As DataSet

Dim myadapter As New OleDb.OleDbDataAdapter("SELECT Student.SNo,
Student.name, Student.surname, Student.sex, Student.FatherName,
Student.MotherName, Student.BirthPlace, Student.BirthDate, Student.school,
Student.phone, Student.Gsm, Student.Email, Student.CityId, Student.Address,
Student.postcode, Student.AdvisorId, Student.OsymNo, Notes FROM Student
Where StudentId=@SID", OleDbConnection1)

myadapter.SelectCommand.Parameters.Add("@SID", SID)

Dim ds As New DataSet

myadapter.Fill(ds)

Return (ds)

End Function

'Function to Update Student


```
<WebMethod()> _
```

```
Public Function Updatestudent(ByVal name As String, ByVal surname As  
String, ByVal Sno As Integer, ByVal school As String, ByVal sex As String,  
ByVal address As String, ByVal cityId As Integer, ByVal postcode As String,  
ByVal gsm As String, ByVal fname As String, ByVal mname As String, ByVal  
bplace As String, ByVal bdate As Date, ByVal OSYM As String, ByVal notes As  
String, ByVal email As String, ByVal phone As String, ByVal SID As Integer)  
As String
```

```
Dim mycommand As New OleDb.OleDbCommand
```

```
mycommand.Connection = OleDbConnection1
```

```
mycommand.CommandText = "UPDATE Student SET Sno=@Sno, name=@name,  
surname=@surname, school=@school, sex=@sex, Address=@address,  
CityId=@city,postcode=@postcode,Gsm=@Gsm,FatherName=@fname,MotherName=@mnam  
e,BirthPlace=@bplace,BirthDate=@bdate,OsymNo=@OSYM,Notes=@notes,Email=@emai  
l,Phone=@phone where StudentId=@SID"
```

```
With mycommand.Parameters
```

```
.Add("@Sno", Sno)
```

```
.Add("@name", name)
```

```
.Add("@surname", surname)
```

```
.Add("@school", school)
```

```
.Add("@sex", sex)
```

```
.Add("@address", address)
```

```
.Add("@city", cityId)
```

```
.Add("@postcode", postcode)
```

```
.Add("@Gsm", gsm)
```

```
.Add("@fname", fname)
```

```
.Add("@mname", mname)
```

```
.Add("@bplace", bplace)
```

```
.Add("@bdate", bdate)
```

```
.Add("@OSYM", OSYM)
```

```
.Add("@notes", notes)
```

```
.Add("@email", email)
```

```
.Add("@phone", phone)
```

```
.Add("@SID", SID)
```

```
End With
```

```
Try
```

```
OleDbConnection1.Open()
```

```
mycommand.ExecuteNonQuery()
```

```
Catch ex As Exception
```

```
Return ex.Message
```

```

    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

'Function to Register Student

```

<WebMethod()> _
Public Function Register_Student(ByVal CourseId As Integer, ByVal SID As
Integer, ByVal BookDate As Date) As String
    Dim mycommand As New OleDb.OleDbCommand
    mycommand.Connection = OleDbConnection1
    mycommand.CommandText = "INSERT INTO
CSR(CourseId,StudentId,BookDate) Values(@CourseId,@StudentId,@BookDate) "
    With mycommand.Parameters
        .Add("@CourseId", CourseId)
        .Add("@StudentId", SID)
        .Add("@BookDate", BookDate)
    End With
    Try
        OleDbConnection1.Open()
        mycommand.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

'Function to List Unregistered Students in Search Form

```

<WebMethod()> _
Public Function SearchUnReg(ByVal DID As Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT Student.SNo,
Student.name, Student.surname, Student.sex, Student.Address, City.CityName,
Student.postcode, Student.Gsm, Student.OsymNo, Student.phone,
Student.Email, Advisor.AdvisorName,Student.StudentId FROM Advisor INNER
JOIN (City INNER JOIN Student ON City.CityId = Student.CityId) ON
Advisor.AdvisorId = Student.AdvisorId where Student.StudentId Not IN
(Select StudentId from CSR) and Student.DepartmentId=@DID",
OleDbConnection1)

```

```

myadapter.SelectCommand.Parameters.Add("@DID", DID)
Dim ds As New DataSet
myadapter.Fill(ds)
Return (ds)
End Function

```

'Function to List Courses that Students are taking

```

<WebMethod()> _
Public Function Find_C_S(ByVal SID As Integer, ByVal DID As Integer) As
DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT
Course.CourseId,Course.CourseName,StartDate,EndDate FROM Course INNER JOIN
CSR ON Course.CourseId = CSR.CourseId WHERE CSR.StudentId=@SID and
DepartmentId=@DID", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@SID", SID)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function

```

'Function to List Deleted Students in Search Form

```

<WebMethod()> _
Public Function SearchDeleted(ByVal DID As Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT Student.SNo,
Student.name, Student.surname, Student.sex, Student.Address, City.CityName,
Student.postcode, Student.Gsm, Student.OsymNo, Student.phone,
Student.Email,
Advisor.AdvisorName,Student.StudentId,Student.DeletionDate,Student.Deletion
Comment FROM Advisor INNER JOIN (City INNER JOIN Student ON City.CityId =
Student.CityId) ON Advisor.AdvisorId = Student.AdvisorId where
Student.isDeleted=true and Student.DepartmentId=@DID", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function

```

'Function to List All Teachers in Search Teacher Form

```

<WebMethod()> _
Public Function ListTeachers(ByVal DID As Integer) As DataSet

```



```

    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT
TeacherId,Tname,Tsurname,Address,Gsm,Phone,Email,Birthdate from Teacher
where isDeleted=false and Teacher.DepartmentId=@DID", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function

```

'Function to List All Teachers who has email in Search Teacher Form

```

<WebMethod()> _
Public Function ListTemails(ByVal DID As Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT
TeacherId,Tname,Tsurname,Address,Gsm,Phone,Email,Birthdate from Teacher
where isDeleted=false and Email<>' ' and Teacher.DepartmentId=@DID",
OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function

```

'Function to List All Teachers who is deleted

```

<WebMethod()> _
Public Function ListTdeleted(ByVal DID As Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT
TeacherId,Tname,Tsurname,Address,Gsm,Phone,Email,Birthdate from Teacher
where isDeleted=true and Teacher.DepartmentId=@DID", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function

```

'Function to List All Teachers who hasn't registered to a course

```

<WebMethod()> _
Public Function ListT_Not_Registered(ByVal DID As Integer) As DataSet

```

```

    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT DISTINCT
Teacher.TeacherId,Teacher.Tname,Teacher.Tsurname,Teacher.Address,Teacher.Gs
m,Teacher.Phone,Teacher.Email,Teacher.Birthdate from Teacher,TCR where
Teacher.isdeleted=false and Teacher.TeacherId NOT IN (Select TCR.TeacherId
from TCR)and Teacher.DepartmentId=@DID", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function

```

'Function to Search Teachers that Teacher Id is taken as input

```

<WebMethod()> _
Public Function Search_T_f_Update(ByVal TID As Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT
Teacher.TeacherId,Teacher.Tname,Teacher.Tsurname,Teacher.Address,Teacher.Gs
m,Teacher.Phone,Teacher.Email,Teacher.Birthdate from Teacher where
Teacher.TeacherId=@TID", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@TID", TID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function

```

'Function to Update Teachers

```

<WebMethod()> _
Public Function Update_Teacher(ByVal name As String, ByVal surname As
String, ByVal address As String, ByVal gsm As String, ByVal phone As
String, ByVal email As String, ByVal bdate As Date, ByVal TID As Integer)
As String
    Dim mycommand As New OleDb.OleDbCommand
    mycommand.Connection = OleDbConnection1
    mycommand.CommandText = "Update Teacher set Tname=@name,
Tsurname=@surname, address=@address, Gsm=@gsm, phone=@phone, Email=@email,
Birthdate=@Bdate where TeacherID=@TID"
    mycommand.Parameters.Add("@name", name)
    mycommand.Parameters.Add("@surname", surname)
    mycommand.Parameters.Add("@address", address)
    mycommand.Parameters.Add("@gsm", gsm)
    mycommand.Parameters.Add("@phone", phone)

```

```

mycommand.Parameters.Add("@email", email)
mycommand.Parameters.Add("@Bdate", bdate)
mycommand.Parameters.Add("@TID", TID)
Try
    OleDbConnection1.Open()
    mycommand.ExecuteNonQuery()
Catch ex As Exception
    Return ex.Message
Finally
    OleDbConnection1.Close()
End Try
End Function

```

'Delete Teacher Function (Actually Update Statements changing the isDelete Field Norecord is deleted physically)

```

<WebMethod()> _
Public Function DelTeacher(ByVal TID As Integer, ByVal DelDate As
DateTime, ByVal DComment As String) As String
    Dim mycommand As New OleDb.OleDbCommand
    mycommand.Connection = OleDbConnection1
    mycommand.CommandText = "UPDATE Teacher set isDeleted=true ,
DelDate=@DelDate, DeletionComment=@DComment where TeacherId=@TID"
    mycommand.Parameters.Add("@DelDate", DelDate)
    mycommand.Parameters.Add("@DComment", DComment)
    mycommand.Parameters.Add("@TID", TID)
    Try
        OleDbConnection1.Open()
        mycommand.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

'Function to Search Teacher for Details Form

```

<WebMethod()> _
Public Function ShowTeacher_Details_personal(ByVal TID As Integer) As
DataSet

```



```

        Dim myadapter As New OleDb.OleDbDataAdapter("SELECT Tname as Name,
Tsurname as Surname, Address, Gsm as Mobile, Phone, Email, BirthDate FROM
Teacher Where TeacherId=@TID", OleDbConnection1)
        myadapter.SelectCommand.Parameters.Add("@TID", TID)
        Dim ds As New DataSet
        myadapter.Fill(ds)
        Return (ds)
    End Function

```

'Function to Return Courses that selected Teacher gives for Details Form

```

    <WebMethod()> _
    Public Function ShowTeacher_Details_course(ByVal TID As Integer) As
DataSet
        Dim myadapter As New OleDb.OleDbDataAdapter("SELECT
CourseTypes.CourseTypeName, Course.CourseCode,Course.CourseName FROM
CourseTypes INNER JOIN (Course INNER JOIN (Teacher INNER JOIN TCR ON
Teacher.TeacherId = TCR.TeacherId) ON Course.CourseId = TCR.CourseId) ON
CourseTypes.CourseTypeId = Course.CourseTypeId where
Teacher.TeacherId=@TID;", OleDbConnection1)
        myadapter.SelectCommand.Parameters.Add("@TID", TID)
        Dim ds As New DataSet
        myadapter.Fill(ds)
        Return (ds)
    End Function

```

'Register Teacher To the Selected Course

```

    <WebMethod()> _
    Public Function Reg_Teacher(ByVal TID As Integer, ByVal CID As Integer)
As String
        Dim mycommand As New OleDb.OleDbCommand
        mycommand.Connection = OleDbConnection1
        mycommand.CommandText = "INSERT INTO TCR(TeacherId,CourseID)
VALUES (@TID,@CID) "
        mycommand.Parameters.Add("@TID", TID)
        mycommand.Parameters.Add("@CID", CID)
        Try
            OleDbConnection1.Open()

```

```

        mycommand.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

'Search a Teacher with his/her Name and Surname in Teacher Search Form

```

<WebMethod()> _
Public Function SearchByNameT(ByVal Name As String, ByVal Surname As
String, ByVal DID As Integer) As DataSet
    Dim myadap As New OleDbDataAdapter("SELECT
TeacherId,Tname,Tsurname,Address,Gsm,Phone,Email,Birthdate from Teacher
WHERE Tname=@Name and Tsurname=@Surname and isDeleted=false and
Teacher.DepartmentId=@DID", OleDbConnection1)
    myadap.SelectCommand.Parameters.Add("@Name", Name)
    myadap.SelectCommand.Parameters.Add("@Surname", Surname)
    myadap.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadap.Fill(ds)
    Return (ds)
End Function

```

'List All Courses that are available for teachers to Register

```

<WebMethod()> _
Public Function Available_Course_Reg() As DataSet
    Dim myadap As New OleDbDataAdapter("Select Distinct Course.CourseId
AS CID,CourseCode+' '+CourseName AS CODE from Course,TCR where
Course.CourseID NOT IN (SELECT TCR.CourseID From TCR)", OleDbConnection1)
    Dim ds As New DataSet
    myadap.Fill(ds)
    Return (ds)
End Function

```

'Search Teacher who gives the specific class that's Taken as Input

```

<WebMethod()> _
Public Function Search_Teacher_ByCourse(ByVal CID As Integer) As DataSet
    Dim myadap As New OleDbDataAdapter("SELECT
Teacher.TeacherId,Tname,Tsurname,Address,Gsm,Phone,Email,Birthdate FROM

```

```

Course INNER JOIN (Teacher INNER JOIN TCR ON Teacher.TeacherId =
TCR.TeacherId) ON Course.CourseId = TCR.CourseId where
Course.CourseId=@CID;", OleDbConnection1)

    myadap.SelectCommand.Parameters.Add("@CID", CID)
    Dim ds As New DataSet
    myadap.Fill(ds)
    Return (ds)
End Function

'Add New Course
<WebMethod()> _
Public Function Add_Course(ByVal CTID As Integer, ByVal name As String,
ByVal Code As String, ByVal Content As String, ByVal Sdate As Date, ByVal
Edate As Date, ByVal Isweek As Boolean, ByVal quota As Integer, ByVal price
As Single, ByVal DID As Integer) As String
    Dim mycommand As New OleDbCommand
    mycommand.Connection = OleDbConnection1
    mycommand.CommandText = "INSERT INTO
Course (CourseTypeId, CourseName, CourseCode, CourseContent, StartDate, EndDate, I
sWeekend, quota, price, DepartmentId)
VALUES (@CTID, @name, @Code, @content, @Sdate, @Edate, @IsWeek, @quota, @price, @DID)
"

    mycommand.Parameters.Add("@CTID", CTID)
    mycommand.Parameters.Add("@name", name)
    mycommand.Parameters.Add("@Code", Code)
    mycommand.Parameters.Add("@Content", Content)
    mycommand.Parameters.Add("@Sdate", Sdate)
    mycommand.Parameters.Add("@Edate", Edate)
    mycommand.Parameters.Add("@Isweek", Isweek)
    mycommand.Parameters.Add("@quota", quota)
    mycommand.Parameters.Add("@price", price)
    mycommand.Parameters.Add("@DID", DID)
    Try
        OleDbConnection1.Open()
        mycommand.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```


'Add New Course Type

<WebMethod()> _

```
Public Function Add_CourseType(ByVal CTname As String, ByVal CTcont As
String) As String
    Dim mycommand As New OleDbCommand
    mycommand.Connection = OleDbConnection1
    mycommand.CommandText = "INSERT INTO
CourseType(CourseTypeName,CourseContent) VALUES(@CTname,@CTcont)"
    mycommand.Parameters.Add("@CTname", CTname)
    mycommand.Parameters.Add("@CTcont", CTcont)
    Try
        OleDbConnection1.Open()
        mycommand.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function
```

'Add New Exam

<WebMethod()> _

```
Public Function Add_Exam(ByVal Edate As Date, ByVal Eplace As String,
ByVal CourseId As Integer, ByVal ETime As DateTime) As String
    Dim mycommand As New OleDbCommand
    mycommand.Connection = OleDbConnection1
    mycommand.CommandText = "INSERT INTO
Exam(ExamDate,ExamPlace,CourseId,ExamTime)
VALUES(@Edate,@Eplace,@CourseId,@Etime)"
    mycommand.Parameters.Add("@Edate", Edate)
    mycommand.Parameters.Add("@Eplace", Eplace)
    mycommand.Parameters.Add("@CourseId", CourseId)
    mycommand.Parameters.Add("@Etime", ETime.TimeOfDay)
    Try
        OleDbConnection1.Open()
        mycommand.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
```

```

        OleDbConnection1.Close()
    End Try
End Function

```

'Search All Advisors who are not dealing with students

```

<WebMethod()> _
Public Function Search_Advisor_NReg(ByVal DID As Integer) As DataSet
    Dim myadap As New OleDbDataAdapter("SELECT Distinct
Advisor.AdvisorId,Advisor.AdvisorName+' '+Advisor.AdvisorSurname As
NameSurname,Advisor.phone,Advisor.Address FROM Advisor, Student WHERE
Advisor.AdvisorId Not In (Select AdvisorId from Student) and
Advisor.DepartmentId=@DID and Advisor.isDeleted=false;", OleDbConnection1)
    myadap.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadap.Fill(ds)
    Return (ds)
End Function

```

'Fuction to list all advisor teachers who are deleted

```

<WebMethod()> _
Public Function List_Del_Advisors(ByVal DID As Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("Select
AdvisorId,AdvisorName+' '+AdvisorSurname As NameSurname,phone,Address from
Advisor where DepartmentId=@DID and isDeleted=true", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function

```

'Fuction to search advisor with name and surname

```

<WebMethod()> _
Public Function Search_Advisors(ByVal DID As Integer, ByVal name As
String, ByVal surname As String) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("Select
AdvisorId,AdvisorName+' '+AdvisorSurname As NameSurname,phone,Address from
Advisor where DepartmentId=@DID and isDeleted=false and AdvisorName=@name
and AdvisorSurname=@surname", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    myadapter.SelectCommand.Parameters.Add("@name", name)

```

```

        myadapter.SelectCommand.Parameters.Add("@surname", surname)
        Dim ds As New DataSet
        myadapter.Fill(ds)
        Return (ds)
End Function

```

'Fuction to search advisor with ID for update Form

```

<WebMethod()> _
Public Function Search_Advisor_ByID(ByVal DID As Integer, ByVal AID As
Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("Select
AdvisorId,AdvisorName,AdvisorSurname,phone,Address from Advisor where
DepartmentId=@DID and isDeleted=false and AdvisorId=@AID",
OleDbConnection1)

    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    myadapter.SelectCommand.Parameters.Add("@AID", AID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function

```

'Fuction to Update Advisor Informations

```

<WebMethod()> _
Public Function Update_Advisor(ByVal name As String, ByVal surname As
String, ByVal phone As String, ByVal address As String, ByVal AID As
Integer) As String
    Dim mycommand As New OleDbCommand
    mycommand.Connection = OleDbConnection1
    mycommand.CommandText = "UPDATE Advisor Set AdvisorName=@name ,
AdvisorSurname=@surname, Phone=@phone, Address=@address where
AdvisorId=@AID"

    mycommand.Parameters.Add("@AdvisorName", name)
    mycommand.Parameters.Add("@AdvisorSurname", surname)
    mycommand.Parameters.Add("@Phone", phone)
    mycommand.Parameters.Add("@Address", address)
    mycommand.Parameters.Add("@AdvisorId", AID)
    Try
        OleDbConnection1.Open()
        mycommand.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    End Try

```



```

        Finally
            OleDbConnection1.Close()
        End Try
    End Function

```

'Fuction to Delete Advisor

```

<WebMethod()> _
Public Function Delete_Advisor(ByVal AID As Integer, ByVal DID As
Integer, ByVal Comment As String, ByVal DelDate As Date) As String
    Dim mycommand As New OleDbCommand
    mycommand.Connection = OleDbConnection1
    mycommand.CommandText = "UPDATE Advisor Set isDeleted=true ,
DeletionComment=@Comment , DelDate=@DelDate where AdvisorId=@AID and
DepartmentId=@DID"
    mycommand.Parameters.Add("@Comment", Comment)
    mycommand.Parameters.Add("@DelDate", DelDate)
    mycommand.Parameters.Add("@AdvisorId", AID)
    mycommand.Parameters.Add("@DID", DID)
    Try
        OleDbConnection1.Open()
        mycommand.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

'Fuction to search students whom a given advisor deals with

```

<WebMethod()> _
Public Function Search_Advisor_Students(ByVal AID As Integer) As
DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT
Student.SNo,Student.name, Student.surname, Student.GSM,Student.birthdate
FROM Advisor INNER JOIN Student ON Advisor.AdvisorId = Student.AdvisorId
where Advisor.AdvisorId=@AID;", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@AID", AID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function

```

'Fuction to List All Exams

<WebMethod()> _

Public Function ListExams(ByVal DID As Integer) As DataSet

```
    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT  
Exam.ExamId, Course.CourseName, Exam.ExamDate, Exam.ExamPlace,  
Exam.ExamTime, Course.CourseId FROM Course INNER JOIN Exam ON  
Course.CourseId = Exam.CourseId where Course.departmentId=@DID",  
OleDbConnection1)
```

```
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
```

```
    Dim ds As New DataSet
```

```
    myadapter.Fill(ds)
```

```
    Return (ds)
```

End Function

'Fuction to Update Exam

<WebMethod()> _

Public Function UpdateExams(ByVal EID As Integer, ByVal Edate As Date,
ByVal Eplace As String, ByVal Etime As DateTime) As String

```
    Dim mycommand As New OleDb.OleDbCommand("UPDATE EXAM SET  
Exam.ExamDate=@Edate, Exam.ExamPlace=@Eplace, Exam.ExamTime=@Etime WHERE  
ExamId=@EID", OleDbConnection1)
```

```
    mycommand.Parameters.Add("@Edate", Edate)
```

```
    mycommand.Parameters.Add("@Eplace", Eplace)
```

```
    mycommand.Parameters.Add("@Etime", Etime)
```

```
    mycommand.Parameters.Add("@EID", EID)
```

```
    Try
```

```
        OleDbConnection1.Open()
```

```
        mycommand.ExecuteNonQuery()
```

```
    Catch ex As Exception
```

```
        Return ex.Message
```

```
    Finally
```

```
        OleDbConnection1.Close()
```

```
    End Try
```

End Function

'Fuction to List All Students Who has entered an exam that's the input

<WebMethod()> _

Public Function ListExamStudents(ByVal CID As Integer, ByVal DID As
Integer) As DataSet

```

    Dim myadapter As New OleDb.OleDbDataAdapter(" SELECT
Student.StudentId,Student.SNo,Student.name FROM Student INNER JOIN ((Course
INNER JOIN Exam ON Course.CourseId = Exam.CourseId) INNER JOIN CSR ON
Course.CourseId = CSR.CourseId) ON Student.StudentId = CSR.StudentId where
Exam.CourseId=@CID and Student.DepartmentId=@DID;", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@CID", CID)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function

```

'Fuction to Add Exam Results to selected Student

```

<WebMethod()> _
Public Function ExamResults(ByVal SID As Integer, ByVal EID As Integer,
ByVal Result As Integer) As String
    Dim mycommand As New OleDbCommand("INSERT INTO
ExamResults(ExamId,StudentId,Result) VALUES(@ExamId,@StudentID,@Result)",
OleDbConnection1)
    With mycommand.Parameters
        .Add("@ExamId", EID)
        .Add("@StudentID", SID)
        .Add("@Result", Result)
    End With
    Try
        OleDbConnection1.Open()
        mycommand.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

'Fuction to List All Users

```

<WebMethod()> _
Public Function ListUsers(ByVal DID As Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter(" SELECT
UserId,UserName,IsAdmin From Login where DepartmentId=@DID",
OleDbConnection1)

```



```

myadapter.SelectCommand.Parameters.Add("@DID", DID)
Dim ds As New DataSet
myadapter.Fill(ds)
Return (ds)
End Function

```

'Fuction to Update Selected User

```
<WebMethod()> _
```

```

Public Function UpdateUser(ByVal DID As Integer, ByVal UID As Integer,
ByVal name As String, ByVal pass As String, ByVal admin As Boolean) As
String

```

'Update User

```

Dim mycommand As New OleDbCommand("UPDATE Login set UserName=@Name,
UserPassword=@Pass, IsAdmin=@Admin WHERE DepartmentId=@DID and
UserId=@UID", OleDbConnection1)

```

```
With mycommand.Parameters
```

```

.Add("@Name", name)
.Add("@Pass", pass)
.Add("@Admin", admin)
.Add("@DID", DID)
.Add("@UID", UID)

```

```
End With
```

```
Try
```

```

OleDbConnection1.Open()
mycommand.ExecuteNonQuery()

```

```
Catch ex As Exception
```

```
Return ex.Message()
```

```
Finally
```

```
OleDbConnection1.Close()
```

```
End Try
```

```
End Function
```

'Fuction to Delete Selected User

```
<WebMethod()> _
```

```
Public Function DeleteUser(ByVal UID As Integer) As String
```

```

Dim mycommand As New OleDbCommand("DELETE FROM LOGIN WHERE
UserId=@UID", OleDbConnection1)

```

```
mycommand.Parameters.Add("@UID", UID)
```

```
Try
```

```

OleDbConnection1.Open()
mycommand.ExecuteNonQuery()

```

```

Catch ex As Exception
    Return ex.Message
Finally
    OleDbConnection1.Close()
End Try
End Function

```

'Fuction to Return Interest Rate

```

<WebMethod()> _
Public Function ReturnRate(ByVal DID As Integer) As Integer
    Dim mycommand As New OleDbCommand("Select InterestRate from
Department Where DepartmentId=@DID", OleDbConnection1)
    mycommand.Parameters.Add("@DID", DID)
    Try
        OleDbConnection1.Open()
        Return mycommand.ExecuteScalar()
    Catch ex As Exception
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

'Fuction to Set Interest Rate for given Department

```

<WebMethod()> _
Public Function SetIRate(ByVal DID As Integer, ByVal Irate As Integer)
As String
    Dim mycommand As New OleDbCommand("UPDATE Department SET
InterestRate=@Irate Where DepartmentId=@DID", OleDbConnection1)
    mycommand.Parameters.Add("@Irate", Irate)
    mycommand.Parameters.Add("@DID", DID)
    Try
        OleDbConnection1.Open()
        mycommand.ExecuteNonQuery()
    Catch ex As Exception
        Return ex.Message
    Finally
        OleDbConnection1.Close()
    End Try
End Function

```

'Fuction to List All installments for a given student and course

<WebMethod()> _

Public Function ListInstallments(ByVal DID As Integer, ByVal SID As Integer, ByVal CID As Integer) As DataSet

Dim myadapter As New OleDb.OleDbDataAdapter("SELECT
Installment.InstallmentId, Course.CourseName, Installment.PaymentDate,
Installment.Price, Installment.IsPaid FROM Department INNER JOIN ((Student
INNER JOIN ((Course INNER JOIN CSR ON Course.CourseId = CSR.CourseId) INNER
JOIN PaySchedule ON Course.CourseId = PaySchedule.CourseId) ON
(Student.StudentId = PaySchedule.StudentId) AND (Student.StudentId =
CSR.StudentId)) INNER JOIN Installment ON PaySchedule.PaymentId =
Installment.PaymentId) ON (Student.DepartmentId = Department.DepartmentId)
AND (Department.DepartmentId = Course.DepartmentId) where
Department.DepartmentId=@DID and Student.StudentId=@SID and
Course.CourseId=@CID", OleDbConnection1)

myadapter.SelectCommand.Parameters.Add("@DID", DID)

myadapter.SelectCommand.Parameters.Add("@SID", SID)

myadapter.SelectCommand.Parameters.Add("@CID", CID)

Dim ds As New DataSet

myadapter.Fill(ds)

Return (ds)

End Function

'Fuction to Update Paid Installment

<WebMethod()> _

Public Function PayInstallment(ByVal IID As Integer) As String

Dim mycommand As New OleDbCommand("UPDATE Installment SET
IsPaid=True Where InstallmentID=@IID", OleDbConnection1)

mycommand.Parameters.Add("@IID", IID)

Try

OleDbConnection1.Open()

mycommand.ExecuteNonQuery()

Catch ex As Exception

Return ex.Message

Finally

OleDbConnection1.Close()

End Try

End Function

'Fuction to List all Payments of a Student


```
<WebMethod()> _
```

```
Public Function ListPayments(ByVal SID As Integer) As DataSet
```

```
    Dim myadapter As New OleDbDataAdapter("SELECT Course.CourseName AS  
COURSE, PaySchedule.Total AS TOTAL, PaySchedule.InAdvanced,  
PaySchedule.NoOfInstallment AS INSTALLMENTS FROM Student INNER JOIN (Course  
INNER JOIN PaySchedule ON Course.CourseId = PaySchedule.CourseId) ON  
Student.StudentId = PaySchedule.StudentId WHERE  
((([Student].[StudentId])=[@SID]))";", OleDbConnection1)
```

```
    myadapter.SelectCommand.Parameters.Add("@SID", SID)
```

```
    Dim ds As New DataSet
```

```
    myadapter.Fill(ds)
```

```
    Return ds
```

```
End Function
```

'List Exam Results of Students For a Given Course

```
<WebMethod()> _
```

```
Public Function ListExamResults(ByVal CID As Integer) As DataSet
```

```
    Dim myadapter As New OleDbDataAdapter("SELECT  
Student.SNo, Student.Name, Student.Surname, ExamResults.Result FROM Student  
INNER JOIN (Course INNER JOIN (Exam INNER JOIN ExamResults ON Exam.ExamId =  
ExamResults.ExamId) ON Course.CourseId = Exam.CourseId) ON  
Student.StudentId = ExamResults.StudentId where Course.CourseId=@CID;",  
OleDbConnection1)
```

```
    myadapter.SelectCommand.Parameters.Add("@CID", CID)
```

```
    Dim ds As New DataSet
```

```
    myadapter.Fill(ds)
```

```
    Return ds
```

```
End Function
```

'List Exam Results of Students For a Given Course

```
<WebMethod()> _
```

```
Public Function ListExamResultsReport(ByVal CID As Integer) As  
DataSet
```

```
    Dim myadapter As New OleDbDataAdapter("SELECT  
Student.SNo, Student.Name, Student.Surname, Course.CourseName, ExamResults.Resu  
lt FROM Student INNER JOIN (Course INNER JOIN (Exam INNER JOIN ExamResults  
ON Exam.ExamId = ExamResults.ExamId) ON Course.CourseId = Exam.CourseId) ON
```

```
Student.StudentId = ExamResults.StudentId where Course.CourseId=@CID;",
OleDbConnection1)
```

```
myadapter.SelectCommand.Parameters.Add("@CID", CID)
```

```
Dim ds As New DataSet
```

```
myadapter.Fill(ds)
```

```
Return ds
```

```
End Function
```

'List Courses of Students For

```
<WebMethod()> _
```

```
Public Function ListCoursesOfStudents(ByVal SID As Integer) As
DataSet
```

```
Dim myadapter As New OleDbDataAdapter("SELECT
Course.CourseId,Course.CourseName FROM Student INNER JOIN (Course INNER
JOIN CSR ON Course.CourseId = CSR.CourseId) ON Student.StudentId =
CSR.StudentId Where Student.StudentId=@SID;", OleDbConnection1)
```

```
myadapter.SelectCommand.Parameters.Add("@SID", SID)
```

```
Dim ds As New DataSet
```

```
myadapter.Fill(ds)
```

```
Return ds
```

```
End Function
```

**'Fuction to List All installments for a given student and course for
detail form**

```
<WebMethod()> _
```

```
Public Function ListInstallmentsForDetail(ByVal DID As Integer, ByVal
SID As Integer, ByVal CID As Integer) As DataSet
```

```
Dim myadapter As New OleDb.OleDbDataAdapter("SELECT
Installment.PaymentDate, Installment.Price, Installment.IsPaid FROM
Department INNER JOIN ((Student INNER JOIN ((Course INNER JOIN CSR ON
Course.CourseId = CSR.CourseId) INNER JOIN PaySchedule ON Course.CourseId =
PaySchedule.CourseId) ON (Student.StudentId = PaySchedule.StudentId) AND
(Student.StudentId = CSR.StudentId)) INNER JOIN Installment ON
PaySchedule.PaymentId = Installment.PaymentId) ON (Student.DepartmentId =
Department.DepartmentId) AND (Department.DepartmentId =
Course.DepartmentId) where Department.DepartmentId=@DID and
Student.StudentId=@SID and Course.CourseId=@CID", OleDbConnection1)
```

```
myadapter.SelectCommand.Parameters.Add("@DID", DID)
```

```
myadapter.SelectCommand.Parameters.Add("@SID", SID)
```

```

myadapter.SelectCommand.Parameters.Add("@CID", CID)
Dim ds As New DataSet
myadapter.Fill(ds)
Return (ds)
End Function

```

'Fuction to List All installments for a given student and course for detail form

```

<WebMethod()> _
Public Function ListInstallmentsForReport(ByVal DID As Integer, ByVal
SID As Integer, ByVal CID As Integer) As DataSet
    Dim myadapter As New OleDb.OleDbDataAdapter("SELECT
Installment.PaymentDate, Installment.Price,
Installment.IsPaid, Student.Name, Student.Surname FROM Department INNER JOIN
((Student INNER JOIN ((Course INNER JOIN CSR ON Course.CourseId =
CSR.CourseId) INNER JOIN PaySchedule ON Course.CourseId =
PaySchedule.CourseId) ON (Student.StudentId = PaySchedule.StudentId) AND
(Student.StudentId = CSR.StudentId)) INNER JOIN Installment ON
PaySchedule.PaymentId = Installment.PaymentId) ON (Student.DepartmentId =
Department.DepartmentId) AND (Department.DepartmentId =
Course.DepartmentId) where Department.DepartmentId=@DID and
Student.StudentId=@SID and Course.CourseId=@CID", OleDbConnection1)
    myadapter.SelectCommand.Parameters.Add("@DID", DID)
    myadapter.SelectCommand.Parameters.Add("@SID", SID)
    myadapter.SelectCommand.Parameters.Add("@CID", CID)
    Dim ds As New DataSet
    myadapter.Fill(ds)
    Return (ds)
End Function
End Class

```


APPENDIX B: DATABASE RELATIONSHIPS

