VB.NET TUTORIAL

.NETDefined

Before getting deeply into the subject we will first know how Businesses are related to Internet, what .NET means to them and what exactly .NET is built upon. As per the product documentation from a Business perspective, there are three phases of the Internet. The First phase gets back to the early 1990's when Internet first came into general use and which brought a big revolution for Businesses. In the First phase of the Internet Businesses designed and launched their Website's and focused on the number of hits to know how many customers were visiting their site and interested in their products, etc. The Second phase is what we are in right now and in this phase Businesses are generating revenue through Online Transactions. We are now moving into the Third phase of the Internet where profit is the main priority. The focus here is to Businesses effectively communicate with their customers and partners who are geographically isolated, participate in Digital Economy and deliver a wide range of services. How can that be possible? The answer, with .NET.

What is .NET ?

Many people reckon that it's Microsoft's way of controlling the Internet, which is false. .NET is Microsoft's strategy of software that provides services to people any time, any place, on any device. An accurate definition of .NET is, it's an XML Web Services platform which allows us to build rich .NET applications, which allows users to interact with the Internet using wide range of smart devices (tablet devices, pocket PC's, web phones etc), which allows to build and integrate Web Services and which comes with many rich set of tools like Visual Studio to fully develop and build those applications.

What are Web Services?

Web Services are the applications that run on a Web Server and communicate with other applications. It uses a series of protocols to respond to different requests. The protocols on which Web Services are built are summarized below:

UDDI: Stands for Universal Discovery and Description Integration. It's said to be the Yellow Pages of Web Services which allows Businesses to search for other Businesses allowing them to search for the services it needs, know about the services and contact them.

WSDL: Stands for Web Services Description Language, often called as whiz-dull. WSDL is an XML document that describes a set of SOAP messages and how those messages are exchanged.

SOAP: Stands for Simple Object Access Protocol. It's the communication protocol for Web Services.

XML, **HTTP** and **SMTP**: Stands for Extensible Markup Language, Hyper Text Transfer Protocol and Simple Message Transfer Protocol respectively. UDDI, WSDL and SOAP rely on these protocols for communication.

The image below shows the order of the protocols on which Web Services are built:

Example of a Web Services Application

Let's say a customer accesses a Website and buys something. The Web services of the business will communicate with the inventory system to see if there is enough stock to fulfill the order. If not, the system can communicate with the suppliers to find one or all of the parts that make up the order before filling the order. At all stages the customer will be kept informed via messages. The end result is a seamless system communicating and exchanging information easily regardless of the platform they are all running on. The business don't need to worry about going to the wrong supplier because it asks the Web service running on the supplier system what it does. And the business doesn't have to worry about the other system's methods of handling data because they communicate via SOAP and XML.

Real World Application

Microsoft's passport service is an example of a .NET service. Passport is a Web-based service designed to make signing in to Websites fast and easy. Passport enables participating sites to authenticate a user with a single set of sign-in credentials eliminating the need for users to remember numerous passwords and sign-in names. You can use one name and password to sign in to all .NET Passport-participating sites and services. You can store personal information in your .NET Passport profile and, if you choose, automatically share that information when you sign in so that participating sites can provide you with personalized services. If you use Hotmail for your email needs then you should be very much familiar with the passport service.

To find out more about how Businesses are implementing Web Services and the advantages it is providing please visit Microsoft's Website and check out the case studies published.

What is .NET Built On?

.NET is built on the Windows Server System to take major advantage of the OS and which comes with a host of different servers which allows for building, deploying, managing and maintaining Web-based solutions. The Windows Server System is designed with performance as priority and it provides scalability, reliability, and manageability for the global, Web-enabled enterprise. The Windows Server System integrated software products are built for interoperability using open Web standards such as XML and SOAP.

Core Windows Server System Products include :

SQL Server2000: This Database Server is Web enabled and is designed with priority for .NET based applications. It is scalable, easy to manage and has a native XML store.

Application Center 2000: This product is designed to manage Web Applications.

Commerce Server 2000: This powerful Server is designed for creating E-Commerce based applications.

Mobile Information Server: This Server provides real-time access for the mobile community. Now Outlook users can use their Pocket PC's to access all their Outlook data while they are moving.

Exchange Server 2000: This is a messaging system Server and allows applications on any device to access information and collaborate using XML.

BizTalk Server 2000: This is the first product created for .NET which is XML based and allows to build business process that integrate with other services in the organization or with other Businesses.

Internet Security and Acceleration Server 2000: This Server provides Security and Protection for machines. It is an integrated firewall and Web cache server built to make the Web-enabled enterprise safer, faster, and more manageable.

Host Integration Server 2000: This Server allows for the Integration of mainframe systems with .NET.

When developing real world projects if you don't know how to use the above mentioned Server's which are built for .NET based applications do not worry. Your System Administrator is always there to help you.

Host Integration Server 2000: This Server allows for the Integration of mainframe systems with .NET.

When developing real world projects if you don't know how to use the above mentioned Server's which are built for .NET based applications do not worry. Your System Administrator is always there to help you.

.NET and XML

There is a lot of connection between XML and .NET. XML is the glue that holds .NET together. XML looks similar to HTML which is readable and text-based. XML is a method of putting structured data into a text file. XML is the specification for defining the structure of the document. Around this specification a whole family of optional modules are being developed. The reason why XML is linked so much to .NET is, it's platform independent and is well supported on any environment. To move the data contained in an XML file around different organizations using different software on different platforms it should be packed it into something. That something is a protocol like SOAP.

About SOAP

SOAP, Simple Object Access Protocol is a simple, lightweight protocol for exchanging information between peers in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelop that describes what is in the message and how it should be processed, a set of encoding rules and a convention for representing remoteprocedure calls and responses.

.NET vs Java

Many of us wonder what .NET has to do with Java. Is there any relation between them? Are they similar? and so on. I even hear some people say .NET is Microsoft's answer to Java. I think every language has its own pros and cons. Java is one of the greatest programming languages created by humans. Java doesn't have a visual interface and requires us to write heaps of code to develop applications. On the other hand, with .NET, the Framework supports around 20 different programming languages which are better and focus only on business logic leaving all other aspects to the Framework. Visual Studio .NET comes with a rich visual interface and supports drag and drop. Many applications were developed, tested and maintained to compare the differences between .NET and Java and the end result was a particular application developed using .NET requires less lines of code, less time to develop and lower deployment costs along with other important issues. Personally, I don't mean to say that Java is gone or .NET based applications are going to dominate the Internet but I think .NET definitely has an extra edge as it is packed with features that simplify application development.

I hope the information above puts some light on the technology aspects behind .NET and helps you in getting started.

.NET Framework

.NET is a "Software Platform". It is a language-neutral environment for developing rich .NET experiences and building applications that can easily and securely operate within it. When developed applications are deployed, those applications will target .NET and will execute wherever .NET is implemented instead of targeting a particular Hardware/OS combination. The components that make up the .NET platform are collectively called the .NET Framework.

The .NET Framework is a managed, type-safe environment for developing and executing applications. The .NET Framework manages all aspects of program execution, like,

allocation of memory for the storage of data and instructions, granting and denying permissions to the application, managing execution of the application and reallocation of memory for resources that are not needed.

The .NET Framework is designed for cross-language compatibility. Cross-language compatibility means, an application written in Visual Basic .NET may reference a DLL file written in C# (C-Sharp). A Visual Basic .NET class might be derived from a C# class or vice versa.

The .NET Framework consists of two main components:

- Common Language Runtime (CLR)
- Class Libraries

Common Language Runtime (CLR)

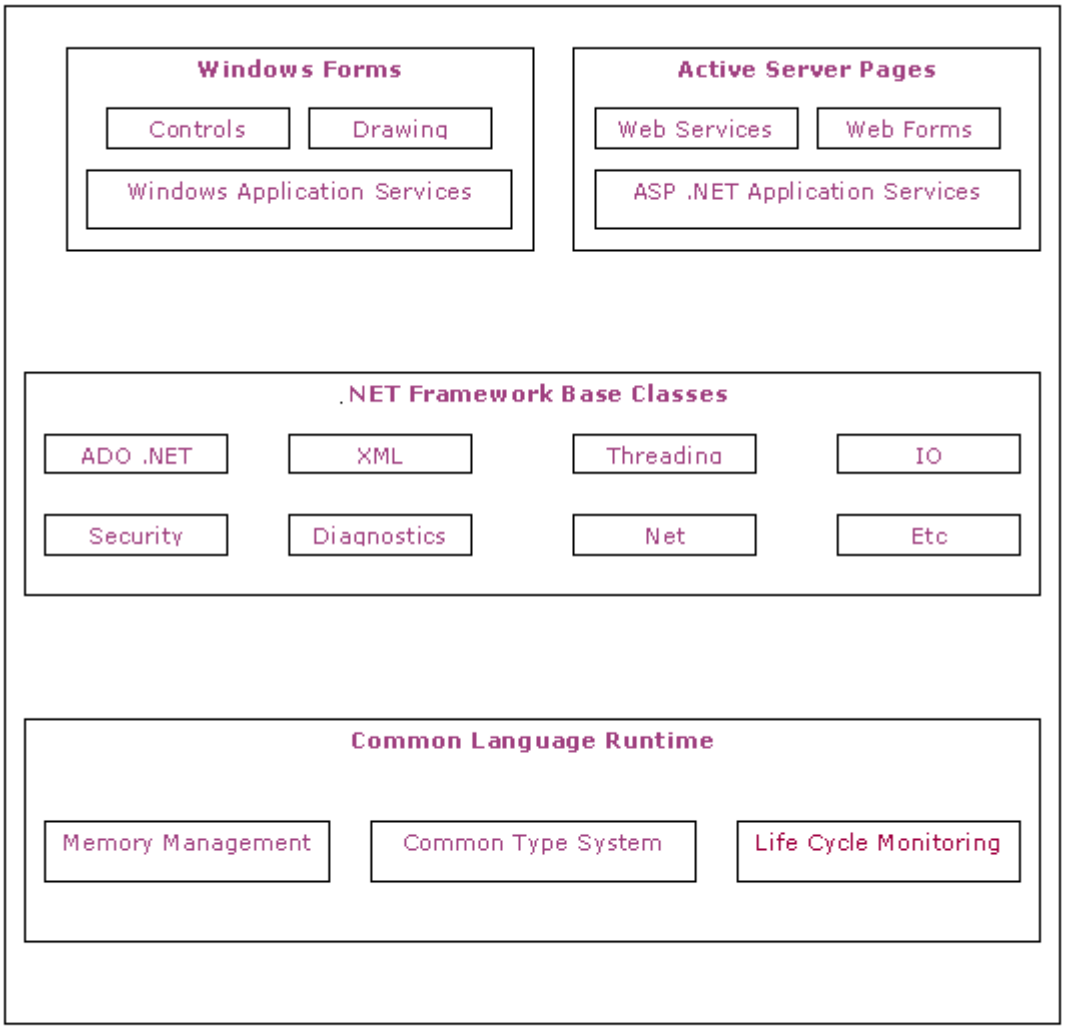
The CLR is described as the "execution engine" of .NET. It provides the environment within which the programs run. It's this CLR that manages the execution of programs and provides core services, such as code compilation, memory allocation, thread management, and garbage collection. Through the Common Type System (CTS), it enforces strict type safety, and it ensures that the code is executed in a safe environment by enforcing code access security. The software version of .NET is actually the CLR version.

Working of the CLR

When the .NET program is compiled, the output of the compiler is not an executable file but a file that contains a special type of code called the Microsoft Intermediate Language(MSIL), which is a low-level set of instructions understood by the common language run time. This MSIL defines a set of portable instructions that are independent of any specific CPU. It's the job of the CLR to translate this Intermediate code into a executable code when the program is executed making the program to run in any environment for which the CLR is implemented. And that's how the .NET Framework achieves Portability. This MSIL is turned into executable code using a JIT (Just In Time) complier. The process goes like this, when .NET programs are executed, the CLR activates the JIT complier. The JIT complier converts MSIL into native code on a demand basis as each part of the program is needed. Thus the program executes as a native code even though it is compiled into MSIL making the program to run as fast as it would if it is compiled to native code but achieves the portability benefits of MSIL.

Class Libraries

Class library is the second major entity of the .NET Framework which is designed to integrate with the common language runtime. This library gives the program access to runtime environment. The class library consists of lots of prewritten code that all the applications created in VB .NET and Visual Studio .NET will use. The code for all the elements like forms, controls and the rest in VB .NET applications actually comes from the class library.



.NET Framework

Common Language Specification (CLS)

If we want the code which we write in a language to be used by programs in other languages then it should adhere to the Common Language Specification (CLS). The CLS describes a set of features that different languages have in common. The CLS defines the minimum standards that .NET language compilers must conform to, and ensures that any source codecompiled by a .NET compiler can interoperate with the .NET Framework.

Some reasons why developers are building applications using the .NET Framework:

- Improved Reliability
- Increased Performance
- Developer Productivity
- Powerful Security
- Integration with existing Systems
- Ease of Deployment
- Mobility Support
- XML Web service Support

- Support for over 20 Programming Languages
- Flexible Data Access

Minimum System Requirements to Install and Use Visual Studio .NET

The minimum requirements are:
RAM: 256 MB (Recommended)
Processor: Pentium II 450 MHz
Operating System: Windows 2000 or Windows XP
Hard Disk Space: 3.5 GB (Includes 500 MB free space on disk)

.NET Framework Advantages

The .NET Framework offers a number of advantages to developers. The following paragraphs describe them in detail.

Consistent Programming Model

Different programming languages have different approaches for doing a task. For example, accessing data with a VB 6.0 application and a VC++ application is totally different. When using different programming languages to do a task, a disparity exists among the approach developers use to perform the task. The difference in techniques comes from how different languages interact with the underlying system that applications rely on.

With .NET, for example, accessing data with a VB .NET and a C# .NET looks very similar apart from slight syntactical differences. Both the programs need to import the System.Data namespace, both the programs establish a connection with the database and both the programs run a query and display the data on a data grid. The VB 6.0 and VC++ example mentioned in the first paragraph explains that there is more than one way to do a particular task within the same language. The .NET example explains that there's a unified means of accomplishing the same task by using the .NET Class Library, a key component of the .NET Framework.

The functionality that the .NET Class Library provides is available to all .NET languages resulting in a consistent object model regardless of the programming language the developer uses.

Direct Support for Security

Developing an application that resides on a local machine and uses local resources is easy. In this scenario, security isn't an issue as all the resources are available and accessed locally. Consider an application that accesses data on a remote machine or has to perform a privileged task on behalf of a nonprivileged user. In this scenario security is much more important as the application is accessing data from a remote machine.

With .NET, the Framework enables the developer and the system administrator to specify method level security. It uses industry-standard protocols such as TCP/IP, XML, SOAP and HTTP to facilitate distributed application communications. This makes distributed computing more secure because .NET developers cooperate with network security devices instead of working around their security limitations.

Simplified Development Efforts

Let's take a look at this with Web applications. With classic ASP, when a developer needs to present data from a database in a Web page, he is required to write the application logic (code) and presentation logic (design) in the same file. He was required to mix the ASP code with the HTML code to get the desired result.

ASP.NET and the .NET Framework simplify development by separating the application logic and presentation logic making it easier to maintain the code. You write the design code (presentation logic) and the actual code (application logic) separately eliminating the need to mix HTML code with ASP code. ASP.NET can also handle the details of maintaining the state of the controls, such as contents in a textbox, between calls to the same ASP.NET page.

Another advantage of creating applications is debugging. Visual Studio .NET and other third party providers provide several debugging tools that simplify application development. The .NET Framework simplifies debugging with support for Runtime diagnostics. Runtime diagnostics helps you to track down bugs and also helps you to determine how well an applicationperforms. The .NET Framework provides three types of Runtime diagnostics: Event Logging, Performance Counters and Tracing.

Easy Application Deployment and Maintenance

The .NET Framework makes it easy to deploy applications. In the most common form, to install an application, all you need to do is copy the application along with the components it requires into a directory on the target computer. The .NET Framework handles the details of locating and loading the components an application needs, even if several versions of the same application exist on the target computer. The .NET Framework ensures that all the components the application depends on are available on the computer before the applicationbegins to execute.

.NET Framework and Languages

As mentioned on the .NET Framework page, .NET Framework is designed for cross-language compatibility.
Cross-language compatibility means .NET components can interact with each other irrespective of the languages they are written in. An application written in VB .NET can reference a DLL file written in C# or a C# application can refer to a resource written in VC++, etc. This language interoperability extends to Object-Oriented inheritance.

This cross-language compatibility is possible due to common language runtime. As you read on the .NET Framework page, when the .NET program is compiled, the output of the compiler is not an executable file but a file that contains a special type of code called the Microsoft Intermediate Language (MSIL). This MSIL is a low-level language which is designed to be read and understood by the common language runtime. Because all .NET executables exist as IL, they can freely operate. The Common Language Specification defines the minimum standards that .NET language compliers must confirm to. Thus, any code compiled by a .NET complier can interoperate with the .NET Framework.

The Common Type System (CTS) defines the rules concerning data types and ensures that code is executed in a safe environment. Since all .NET applications are converted to IL before execution all primitive data types are represented as .NET types. This means that, a VB Integer and a C# int are both represented in IL code as System.Int32. Because both the languages use a common and interconvertible type system, it is possible to transfer data between components and avoid time-consuming conversions.

Languages supported by .NET Framework

The table below lists all the languages supported by the .NET Framework and describes those languages. The languages listed below are supported by the .NET Framework upto the year 2003. In future there may be other languages that the .NET Framework might support.

Language	Description/Usage
APL	APL is one of the most powerful, consistent and concise computer programming languages ever devised. It is a language for describing procedures in the processing of information. It can be used to describe mathematical procedures having nothing to do with computers or to describe the way a computer works.
C++	C++ is a true OOP. It is one of the early Object-Oriented programming languages. C++ derives from the C language. **VC++** Visual C++ is the name of a C++ compiler with an integrated environment from Microsoft. This includes special tools that simplify the development of great applications, as well as specific libraries. Its use is known as visual programming.
C#	C# called as C Sharp is a full fledged Object-Oriented programming language from Microsoft built into the .NET Framework. First created in the late 1990's was part of Microsoft's whole .NET strategy.
Cobol	COBOL (Common Business Oriented Language) was the first widely-used high-level programming language for businessapplications. It is considered as a programming language to have more lines of code than any other language.
Component Pascal	Component Pascal is a Pascal derived programming language that is specifically designed for programming software components.
Curriculum	No information.
Eiffel	Eiffel is an Object-Oriented (OO) programming language which emphasizes the production of robust software. Eiffel is strongly statically typed mature Object-Oriented language with automatic memory management.
Forth	Forth is a programming language and programming

	environment. It features both interactive execution of commands (making it suitable as a shell for systems that lack a more formal operating system), as well as the ability to compile sequences of commands into threaded code for later execution.
Fortran	Acronym for Formula Translator, Fortran is one of the oldest high-level programming languages that is still widely used in scientific computing because of its compact notation for equations, ease in handling large arrays, and huge selection of library routines for solving mathematical problems efficiently.
Haskell	Haskell is a computer programming language that is a polymorphicly typed, lazy, purely functional language, quite different from most other programming languages. It is a wide-spectrum language, suitable for a variety of applications. It is particularly suitable for programs which need to be highly modifiable and maintainable.
Java Language	The Java language is one of the most powerful Object-Oriented programming languages developed till date. It's platform independence (not depending on a particular OS) feature makes it a very popular programming language.
Microsoft JScript	Microsoft JScript is the Microsoft implementation of the ECMA 262 language specification. JScript is an interpreted, object-based scripting language. It has fewer capabilities than full-fledged Object-Oriented languages like C++ but is more than sufficiently powerful for its intended purposes.
Mercury	Mercury is a new logic/functional programming language, which combines the clarity and expressiveness of declarative programming with advanced static analysis and error detection features. Its highly optimized execution algorithm delivers efficiency far in excess of existing logic programming systems, and close to conventional programming systems. Mercury addresses the problems of large-scale program development, allowing modularity, separate compilation, and numerous optimization/time trade-offs.
Mondrian	Mondrian is a simple functional scripting language for Internet applications. It is a functional language specifically designed to inter-operate with other languages in an OO environment. Current versions of Mondrian run on .NET. Mondrian also supports ASP.NET, allowing you to embed functional language code in web pages along with C# code.
Oberon	Oberon is a programming language very much like Modula-2 in syntax but with several interesting features. It's based on OOP concepts and provides a Windows-based graphical user

	interface.
Oz	Oz is a high-level programming language that combines constraint inference with concurrency. Oz is dynamically typed and has first-class procedures, classes, objects, exceptions and sequential threads synchronizing over a constraint store. It supports finite domain and feature constraints and has powerful primitives for programming constraint inference engines at a high level.
Pascal	Principle objectives for Pascal were for the language to be efficent to implement and run, allow for the development of well structured and well organized programs, and to serve as a vehicle for the teaching of the important concepts of computer programming. The Prime area of application that Pascal entails is the learning environment. This language was not really developed to be used for anything other than teaching students the basics of programming as it was originally developed for this purpose.
Perl	Practical Extraction and Report Language, Perl, is a language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks.
Python	Python is an interpreted, interactive, Object-Oriented programming language. Python combines remarkable power with very clear syntax. It has modules, classes, exceptions, very high level dynamic data types, and dynamic typing.
RPG	Report Program Generator, RPG, is used for generation of reports from data files, including matching record and sub-total reports. RPG is one of the few languages created for punch card machines that is still in common use today. RPG or RPG IV is a native programming language for IBM's iSeries minicomputer system.
Scheme	Scheme is a statically scoped programming language. It was designed to have an exceptionally clear and simple semantics and few different ways to form expressions. A wide variety of programming paradigms, including imperative, functional, and message passing styles, find convenient expression in Scheme.
Small Talk	SmallTalk is an expressive language that uses a simple sub set of human languages, nouns and verbs. Smalltalk was the first, and remains one of the few, pure object systems, which simply means that everything in a Smalltalk program is an object. Smalltalk is generally recognized as the second Object Programming Language (OPL).

Standard ML	Standard ML is a safe, modular, strict, functional, polymorphic programming language with compile-time type checking and type inference, garbage collection, exception handling, immutable data types and updatable references, abstract data types, and parametric modules. It has efficient implementations and a formal definition with a proof of soundness.
Microsoft Visual Basic	Visual Basic is a "visual programming" environment for developing Windows applications. Visual Basic makes it possible to develop complicated applications very quickly. This site is all about Visual Basic.

Visual Basic .NET

Visual Basic .NET provides the easiest, most productive language and tool for rapidly building Windows and Web applications. Visual Basic .NET comes with enhanced visual designers, increased application performance, and a powerful integrated development environment (IDE). It also supports creation of applications for wireless, Internet-enabled hand-held devices. The following are the features of Visual Basic .NET with .NET Framework 1.0 and Visual Basic .NET 2003 with .NET Framework 1.1. This also answers why should I use Visual Basic .NET, what can I do with it?

Powerful Windows-based Applications

Visual Basic .NET comes with features such as a powerful new forms designer, an in-place menu editor, and automatic control anchoring and docking. Visual Basic .NET delivers new productivity features for building more robust applications easily and quickly. With an improved integrated development environment (IDE) and a significantly reduced startup time,Visual Basic .NET offers fast, automatic formatting of code as you type, improved IntelliSense, an enhanced object browser and XML designer, and much more.

Building Web-based Applications

With Visual Basic .NET we can create Web applications using the shared Web Forms Designer and the familiar "drag and drop" feature. You can double-click and write code to respond to events. Visual Basic .NET 2003 comes with an enhanced HTML Editor for working with complex Web pages. We can also use IntelliSense technology and tag completion, or choose the WYSIWYG editor for visual authoring of interactive Web applications.

Simplified Deployment

With Visual Basic .NET we can build applications more rapidly and deploy and maintain them with efficiency. Visual Basic .NET 2003 and .NET Framework 1.1 makes "DLL Hell" a thing of the past. Side-by-side versioning enables multiple versions of the same component to live safely on the same machine so that applications can use a specific version of a component. XCOPY-deployment and Web auto-download of Windows-

based applications combine the simplicity of Web page deployment and maintenance with the power of rich, responsive Windows-based applications.

Powerful, Flexible, Simplified Data Access

You can tackle any data access scenario easily with ADO.NET and ADO data access. The flexibility of ADO.NET enables data binding to any database, as well as classes, collections, and arrays, and provides true XML representation of data. Seamless access to ADO enables simple data access for connected data binding scenarios. Using ADO.NET, Visual Basic .NET can gain high-speed access to MS SQL Server, Oracle, DB2, Microsoft Access, and more.

Improved Coding

You can code faster and more effectively. A multitude of enhancements to the code editor, including enhanced IntelliSense, smart listing of code for greater readability and a background compiler for real-time notification of syntax errors transforms into a rapid application development (RAD) coding machine.

Direct Access to the Platform

Visual Basic developers can have full access to the capabilities available in .NET Framework 1.1. Developers can easily program system services including the event log, performance counters and file system. The new Windows Service project template enables to build real Microsoft Windows NT Services. Programming against Windows Services and creating new Windows Services is not available in Visual Basic .NET Standard, it requires Visual Studio 2003 Professional, or higher.

Full Object-Oriented Constructs

You can create reusable, enterprise-class code using full object-oriented constructs. Language features include full implementation inheritance, encapsulation, and polymorphism. Structured exception handling provides a global error handler and eliminates spaghetti code.

XML Web Services

XML Web services enable you to call components running on any platform using open Internet protocols. Working with XML Web services is easier where enhancements simplify the discovery and consumption of XML Web services that are located within any firewall. XML Web services can be built as easily as you would build any class in Visual Basic 6.0. The XML Web service project template builds all underlying Web service infrastructure.

Mobile Applications

Visual Basic .NET 2003 and the .NET Framework 1.1 offer integrated support for developing mobile Web applications for more than 200 Internet-enabled mobile devices. These new features give developers a single, mobile Web interface and programming model to support a broad range of Web devices, including WML 1.1 for WAP—enabled cellular phones, compact HTML (cHTML) for i-Mode phones, and HTML for Pocket PC, handheld devices,

and pagers. Please note, Pocket PC programming is not available in Visual Basic .NET Standard, it requires Visual Studio 2003 Professional, or higher.

COM Interoperability

You can maintain your existing code without the need to recode. COM interoperability enables you to leverage your existing code assets and offers seamless bi-directional communication between Visual Basic 6.0 and Visual Basic .NET applications.

Reuse Existing Investments

You can reuse all your existing ActiveX Controls. Windows Forms in Visual Basic .NET 2003 provide a robust container for existing ActiveX controls. In addition, full support for existing ADO code and data binding enable a smooth transition to Visual Basic .NET 2003.

Upgrade Wizard

You upgrade your code to receive all of the benefits of Visual Basic .NET 2003. The Visual Basic .NET Upgrade Wizard, available in Visual Basic .NET 2003 Standard Edition, and higher, upgrades up to 95 percent of existing Visual Basic code and forms to Visual Basic .NET with new support for Web classes and UserControls.

OOP with VB

OOP Basics

Visual Basic was Object-Based, Visual Basic .NET is Object-Oriented, which means that it's a true Object-Oriented Programming Language. Visual Basic .NET supports all the key OOP features like Polymorphism, Inheritance, Abstraction and Encapsulation. It's worth having a brief overview of OOP before starting OOP with VB.

Why Object Oriented approach?

A major factor in the invention of Object-Oriented approach is to remove some of the flaws encountered with the procedural approach. In OOP, data is treated as a critical element and does not allow it to flow freely. It bounds data closely to the functions that operate on it and protects it from accidental modification from outside functions. OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects. A major advantage of OOP is code reusability.

Some important features of Object Oriented programming are as follows:

- Emphasis on data rather than procedure
- Programs are divided into Objects
- Data is hidden and cannot be accessed by external functions
- Objects can communicate with each other through functions
- New data and functions can be easily added whenever necessary
- Follows bottom-up approach

Concepts of OOP:

- Objects
- Classes
- Data Abstraction and Encapsulation
- Inheritance
- Polymorphism

Briefly on Concepts:

Objects

Objects are the basic run-time entities in an object-oriented system. Programming problem is analyzed in terms of objects and nature of communication between them. When a program is executed, objects interact with each other by sending messages. Different objects can also interact with each other without knowing the details of their data or code.

Classes

A *class* is a collection of objects of similar type. Once a class is defined, any number of objects can be created which belong to that class.

Data Abstraction and Encapsulation

Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes.
Storing data and functions in a single unit (class) is *encapsulation.* Data cannot be accessible to the outside world and only those functions which are stored in the class can access it.

Inheritance

Inheritance is the process by which objects can acquire the properties of objects of other class. In OOP, *inheritance* provides reusability, like, adding additional features to an existing class without modifying it. This is achieved by deriving a new class from the existing one. The new class will have combined features of both the classes.

Polymorphism

Polymorphism means the ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends on the data types used in the operation. Polymorphism is extensively used in implementing Inheritance.

Advantages of OOP

Object-Oriented Programming has the following advantages over conventional approaches:

- OOP provides a clear modular structure for programs which makes it good for defining abstract datatypes where implementation details are hidden and the unit has a clearly defined interface.

- OOP makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones.
- OOP provides a good framework for code libraries where supplied software components can be easily adapted and modified by the programmer. This is particularly useful for developing graphical user interfaces

OOP with VB

Visual Basic .NET is Object-Oriented. Everything we do in Visual Basic involves objects in some way or other and everything is based on the Object class. Controls, Forms, Modules, etc are all types of classes. Visual Basic .NET comes with thousands of built-in classes which are ready to be used. Let's take a closer look at Object-Oriented Programming in Visual Basic. We will see how we can create classes, objects, how to inherit one class from other, what is polymorphism, how to implement interfaces and so on. We will work with Console Applicationshere as they are simple to code.

Classes and Objects

Classes are types and Objects are instances of the Class. Classes and Objects are very much related to each other. Without objects you can't use a class. In Visual Basic we create a class with the Class statement and end it with End Class. The Syntax for a Class looks as follows:

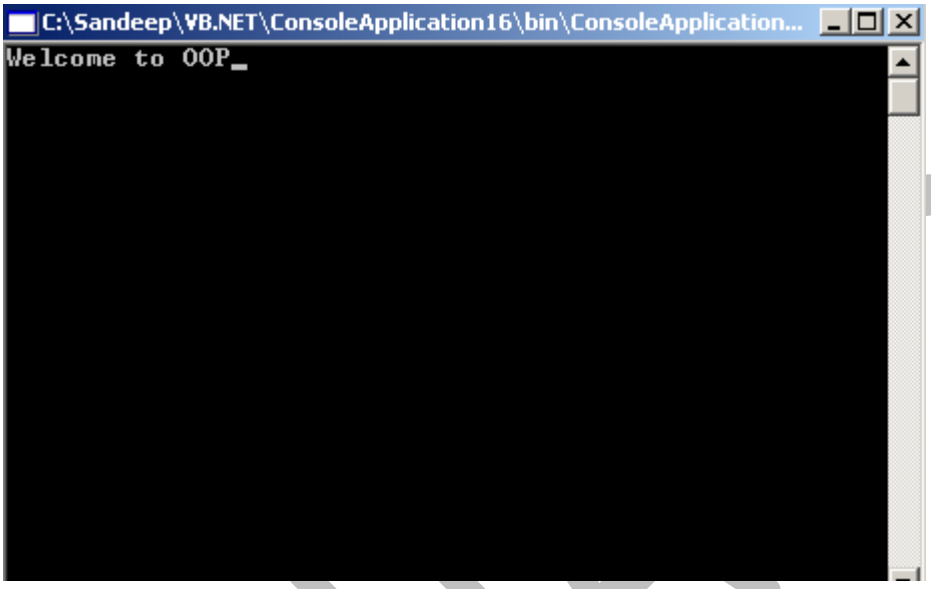
```
Public Class Test

----- Variables
-----Methods
-----Properties
-----Events

End Class
```

The above syntax created a class named Test. To create a object for this class we use the new keyword and that looks like this: **Dim obj as new Test()**. The following code shows how to create a Class and access the class with an Object. Open a Console Application and place the following code in it.

```
Module Module1
Imports System.Console

Sub Main()
Dim obj As New Test()
'creating a object obj for Test class
obj.disp()
'calling the disp method using obj
Read()
End Sub

End Module
```

```
Public Class Test
'creating a class named Test
Sub disp()
'a method named disp in the class
Write("Welcome to OOP")
End Sub
End Class
```

Output of above code is the image below.



Fields, Properties, Methods and Events

Fields, Properties, Methods, and Events are members of the class. They can be declared as Public, Private, Protected, Friend or Protected Friend.

Fields and Properties represent information that an object contains. Fields of a class are like variables and they can be read or set directly. For example, if you have an object named House, you can store the numbers of rooms in it in a field named Rooms. It looks like this:

```
Public Class House
Public Rooms as Integer
End Class
```

Properties are retrieved and set like fields but are implemented using Property Get and Property Set procedures which provide more control on how values are set or returned.

Methods represent the object's built-in procedures. For example, a Class named Country may have methods named Area and Population. You define methods by adding procedures, Sub routines or functions to your class. For example, implementation of the Area and Population methods discussed above might look like this

```
Public Class Country
Public Sub Area()
Write("--------")
End Sub
Public Sub population()
Write("---------")
End Sub
End Class
```

Events allow objects to perform actions whenever a specific occurrence takes place. For example when we click a button a click event occurs and we can handle that event in an eventhandler.

Constructors

A constructor is a special member function whose task is to initialize the objects of it's class. This is the first method that is run when an instance of a type is created. A constructor is invoked whenever an object of it's associated class is created. If a class contains a constructor, then an object created by that class will be initialized automatically. We pass data to the constructor by enclosing it in the parentheses following the class name when creating an object. Constructors can never return a value, and can be overridden to provide custom intitialization functionality. In Visual Basic we create constructors by adding a Sub procedure named New to a class. The following code demonstrates the use of constructors in Visual Basic.

```
Module Module1

Sub Main()

Dim con As New Constructor(10)
WriteLine(con.display())
'storing a value in the constructor by passing a value(10) and calling it with the
'display method
Read()

End Sub

End Module



Public Class Constructor
Public x As Integer

Public Sub New(ByVal value As Integer)
'constructor
x = value
'storing the value of x in constructor
End Sub
```

```
Public Function display() As Integer
Return x
'returning the stored value
End Function

End Class
```

Destructors

A destructor, also know as finalizer, is the last method run by a class. Within a destructor we can place code to clean up the object after it is used, which might include decrementingcounters or releasing resources. We use Finalize method in Visual Basic for this and the Finalize method is called automatically when the .NET runtime determines that the object is no longer required. When working with destructors we need to use the overrides keyword with Finalize method as we will override the Finalize method built into the Object class. We normally use Finalize method to deallocate resources and inform other objects that the current object is going to be destroyed. Because of the nondeterministic nature of garbage collection, it is very hard to determine when a class's destructor will be called. The following code demonstrates the use of Finalize method.

```
Module Module1

Sub Main()
Dim obj As New Destructor()
End Sub

End Module

Public Class Destructor

Protected Overrides Sub Finalize()
Write("hello")
Read()
End Sub

End Class
```

When you run the above code, the word and object, obj of class, destructor is created and "Hello" is displayed. When you close the DOS window, obj is destroyed.

Inheritance

A key feature of OOP is reusability. It's always time saving and useful if we can reuse something that already exists rather than trying to create the same thing again and again. Reusing the class that is tested, debugged and used many times can save us time and effort of developing and testing it again. Once a class has been written and tested, it can be used by other programs to suit the program's requirement. This is done by creating a new class from an existing class. The process of deriving a new class from an existing class is called Inheritance. The old class is called the base class and the new class is called derived class. The derived class inherits some or everything of the base class. In Visual Basic we use

the Inherits keyword to inherit one class from other. The general form of deriving a new class from an existing class looks as follows:

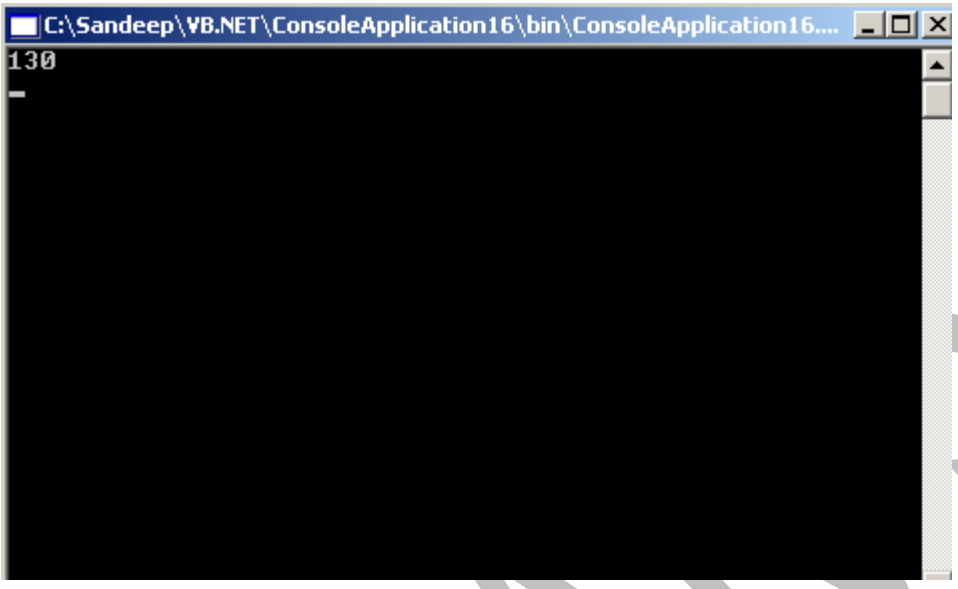
```
Public Class One
---
---
End Class

Public Class Two
    Inherits One
---
---
End Class
```

Using Inheritance we can use the variables, methods, properties, etc, from the base class and add more functionality to it in the derived class. The following code demonstrates the process of Inheritance in Visual Basic.

```
Imports System.Console
Module Module1

Sub Main()
Dim ss As New Two()
WriteLine(ss.sum())
Read()
End Sub

End Module

Public Class One
'base class
Public i As Integer = 10
Public j As Integer = 20

Public Function add() As Integer
Return i + j
End Function

End Class

Public Class Two
   Inherits One
'derived class. class two inherited from class one
Public k As Integer = 100

Public Function sum() As Integer
'using the variables, function from base class and adding more
functionality
Return i + j + k
```

```
End Function

End Class
```

Output of above code is sum of i, j, k as shown in the image below.



Polymorphism

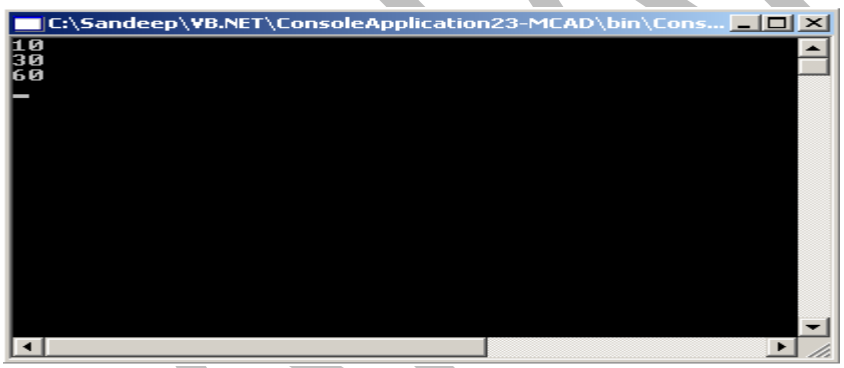
Polymorphism is one of the crucial features of OOP. It means "one name, multiple forms". It is also called as Overloading which means the use of same thing for different purposes. Using Polymorphism we can create as many functions we want with one function name but with different argument list. The function performs different operations based on the argument list in the function call. The exact function to be invoked will be determined by checking the type and number of arguments in the function.

The following code demonstrates the implementation of Polymorphism.

```
Module Module1

Sub Main()
Dim two As New One()
WriteLine(two.add(10))
'calls the function with one argument
WriteLine(two.add(10, 20))
'calls the function with two arguments
WriteLine(two.add(10, 20, 30))
'calls the function with three arguments
Read()
End Sub

End Module
```

```
Public Class One
Public i, j, k As Integer

Public Function add(ByVal i As Integer) As Integer
'function with one argument
Return i
End Function

Public Function add(ByVal i As Integer, ByVal j As Integer) As Integer
'function with two arguments
Return i + j
End Function

Public Function add(ByVal i As Integer, ByVal j As Integer, ByVal k As
Integer) As Integer
'function with three arguments
Return i + j + k
End Function

End Class
```

Output of the above code is shown in the image below.



Interfaces

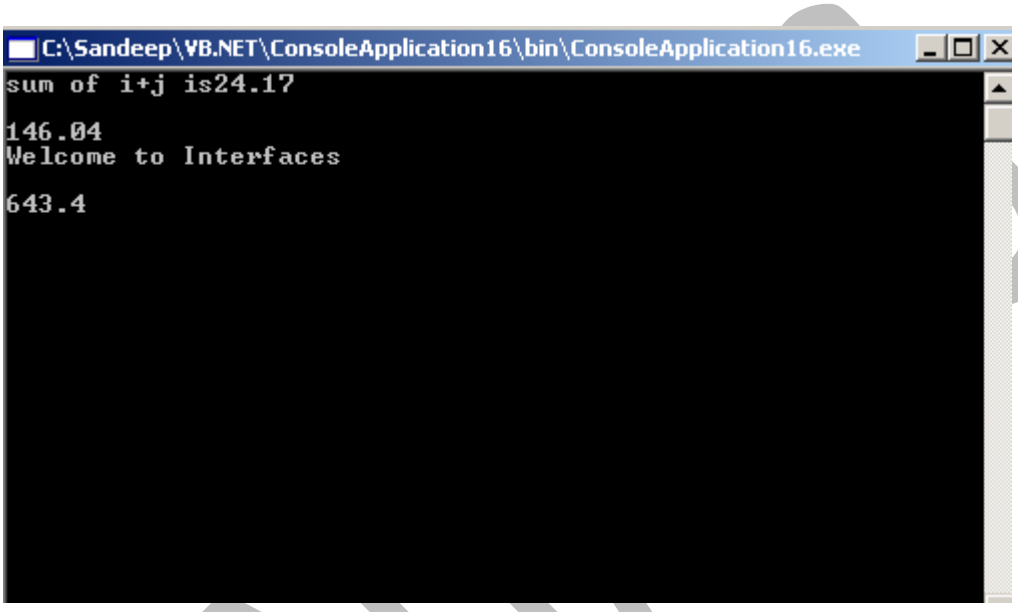
Interfaces allow us to create definitions for component interaction. They also provide another way of implementing polymorphism. Through interfaces, we specify methods that a component must implement without actually specifying how the method is implemented. We just specify the methods in an interface and leave it to the class to implement thosemethods. Visual Basic .NET does not support multiple inheritance directly but using interfaces we can achieve multiple inheritance. We use the Interface keyword to create an interface and implements keyword to implement the interface. Once you create an interface you need to implement all the methods specified in that interface. The following code demonstrates the use of interface.

```
Imports System.Console
Module Module1
```

```
Sub Main()
Dim OneObj As New One()
Dim TwoObj As New Two()
'creating objects of class One and Two
OneObj.disp()
OneObj.multiply()
TwoObj.disp()
TwoObj.multiply()
'accessing the methods from classes as specified in the interface
End Sub

End Module

Public Interface Test
'creating an Interface named Test
Sub disp()
Function Multiply() As Double
'specifying two methods in an interface
End Interface

Public Class One
  Implements Test
'implementing interface in class One

Public i As Double = 12
Public j As Double = 12.17

Sub disp() Implements Test.disp
'implementing the method specified in interface
WriteLine("sum of i+j is" & i + j)
Read()
End Sub

Public Function multiply() As Double Implements Test.Multiply
'implementing the method specified in interface
WriteLine(i * j)
Read()
End Function

End Class

Public Class Two
  Implements Test
'implementing the interface in class Two

Public a As Double = 20
Public b As Double = 32.17

Sub disp() Implements Test.disp
WriteLine("Welcome to Interfaces")
```

```
Read()
End Sub

Public Function multiply() As Double Implements Test.Multiply
WriteLine(a * b)
Read()
End Function

End Class
```

Output of above code is the image below.



Abstract Classes

An abstract class is the one that is not used to create objects. An abstract class is designed to act as a base class (to be inherited by other classes). Abstract class is a design concept inprogram development and provides a base upon which other classes are built. Abstract classes are similar to interfaces. After declaring an abstract class, it cannot be instantiated on it's own, it must be inherited. Like interfaces, abstract classes can specify members that must be implemented in inheriting classes. Unlike interfaces, a class can inherit only one abstractclass. Abstract classes can only specify members that should be implemented by all inheriting classes.

Creating Abstract Classes

In Visual Basic .NET we create an abstract class by using the MustInherit keyword.
An abstract class like all other classes can implement any number of members. Members of anabstract class can either be Overridable (all the inheriting classes can create their own implementation of the members) or they can have a fixed implementation that will be common to all inheriting members. Abstract classes can also specify abstract members. Like abstract classes, abstract members also provide no details regarding their implementation. Only the member type, access level, required parameters and return type are

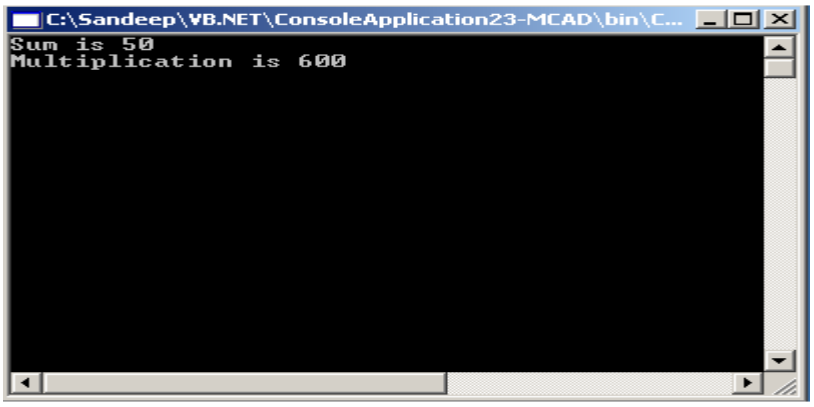
specified. To declare an abstract member we use
the MustOverride keyword. Abstract members should be declared in abstract classes.

Implementing Abstract Class

When a class inherits from an abstract class, it must implement every abstract member defined by the abstract class. Implementation is possible by overriding the member specified in the abstract class. The following code demonstrates the declaration and implementation of an abstract class.

```
Module Module1

Public MustInherit Class AbstractClass
'declaring an abstract class with MustInherit keyword
Public MustOverride Function Add() As Integer
Public MustOverride Function Mul() As Integer
'declaring two abstract members with MustOverride keyword
End Class

Public Class AbstractOne
        Inherits AbstractClass
'implementing the abstract class by inheriting
Dim i As Integer = 20
Dim j As Integer = 30
'declaring two integers

Public Overrides Function Add() As Integer
Return i + j
End Function
'implementing the add method

Public Overrides Function Mul() As Integer
Return i * j
End Function
'implementing the mul method

End Class

Sub Main()
Dim abs As New AbstractOne()
'creating an instance of AbstractOne
WriteLine("Sum is" & " " & abs.Add())
WriteLine("Multiplication is" & " " & abs.Mul())
'displaying output
Read()
End Sub

End Module
```

The output of above code is the image below.



Structures

Structures can be defined as a tool for handling a group of logically related data items. They are user-defined and provide a method for packing together data of different types.Structures are very similar to Classes. Like Classes, they too can contain members such as fields and methods. The main difference between classes and structures is, classes arereference types and structures are value types. In practical terms, structures are used for smaller lightweight objects that do not persist for long and classes are used for larger objects that are expected to exist in memory for long periods. We declare a structure in Visual Basic .NET with the Structure keyword.

Value Types and Reference Types

Value Types and Reference Types belong to Application data memory and the difference between them is the way variable data is accessed. We will have a brief overview about them.

Value Types

In VB .NET we use the Dim statement to create a variable that represents a value type. For example, we declare a integer variable with the following statement: **Dim x as Integer**. The statement tells the run time to allocate the appropriate amount of memory to hold an integer variable. The statement creates a variable but does not assign a value to it. We assign a value to that variable like this: **x=55**. When a variable of value type goes out of scope, it is destroyed and it's memory is reclaimed.

Reference Types

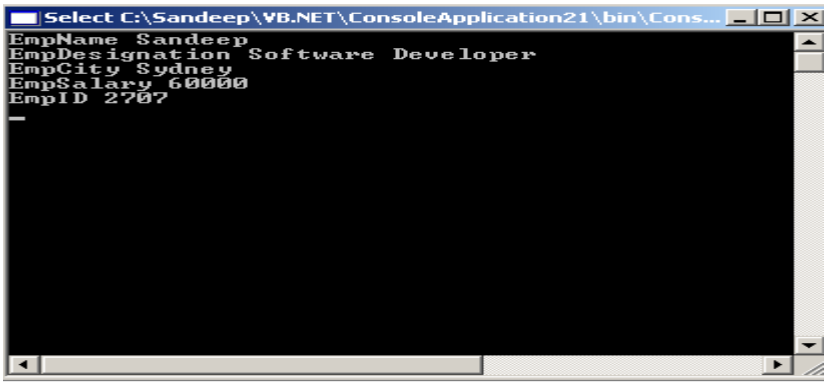
Creating a variable of reference type is a two-step process, declare and instantiate. The first step is to declare a variable as that type. For example, the following statement **Dim Form1 as new System.Windows.Forms.Form** tells the run time to set enough memory to hold a Form variable. The second step, instantiation, creates the object. It looks like this in code:**Form1=New System.Windows.Forms.Form**. A variable of reference type exists in two memory locations and that's why when that variable goes out of scope, the reference to that object is destroyed but the object itself is not destroyed. If any other references to that object exist, the object remains intact. If no references exist to that object then it is subject to garbage collection.

Code for Creating a Structure

The following code creates a Structure named Employee with five fields of different data types.

```
Module Module1

Structure Employee
'declaring a structure named Employee
Dim EmpName As String
Dim EmpDesignation As String
Dim EmpCity As String
Dim EmpSal As Double
Dim EmpId As Integer
'declaring five fields of different data types in the structure
End Structure

Sub Main()
Dim san As New Employee()
'creating an instance of Employee
san.EmpName = "Sandeep"
san.EmpDesignation = "Software Developer"
san.EmpCity = "Sydney"
san.EmpSal = 60000
san.EmpId = 2707
'assigning values to member variables
WriteLine("EmpName" + " " + san.EmpName)
WriteLine("EmpDesignation" + " " + san.EmpDesignation)
WriteLine("EmpCity" + " " + san.EmpCity)
WriteLine("EmpSalary" + " " + san.EmpSal.ToString)
WriteLine("EmpID" + " " + san.EmpId.ToString)
'accessing member variables with the period/dot operator
Read()
End Sub

End Module
```

The output of above code is the image below.



VB Language

Visual Basic, the name makes me feel that it is something special. In the History of Computing world no other product sold more copies than Visual Basic did. Such is the importance of that language which clearly states how widely it is used for developing applications. Visual Basic is very popular for it's friendly working (graphical) environment. Visual Basic. NET is an extension of Visual Basic programming language with many new features in it. The changes from VB to VB .NET are huge, ranging from the change in syntax of the language to the types of projects we can create now and the way we design applications. Visual Basic .NET was designed to take advantage of the .NET Framework base classes and runtime environment. It comes with power packed features that simplify application development.

Briefly on some changes:

- The biggest change from VB to VB .NET is, VB .NET is Object-Oriented now. VB .NET now supports all the key OOP features like Inheritance, Polymorphism, Abstraction and Encapsulation. We can now create classes and objects, derive classes from other classes and so on. The major advantage of OOP is code reusability
- The Command Button now is Button and the TextBox is TextBox instead of Text as in VB6
- Many new controls have been added to the toolbar to make application development more efficient
- VB .NET now adds Console Applications to it apart from Windows and Web Applications. Console applications are console oriented applications that run in the DOS version
- All the built-in VB functionality now is encapsulated in a Namespace (collection of different classes) called System
- New keywords are added and old one's are either removed or renamed
- VB .NET is strongly typed which means that we need to declare all the variables by default before using them
- VB .NET now supports structured exception handling using Try...Catch...Finally syntax
- The syntax for procedures is changed. Get and Let are replaced by Get and Set
- Event handling procedures are now passed only two parameters
- The way we handle data with databases is changed as well. VB .NET now uses ADO .NET, a new data handling model to communicate with databases on local machines or on a network and also it makes handling of data on the Internet easy. All the data in ADO .NET is represented in XML format and is exchanged in the same format. Representing data in XML format allows us for sending large amounts of data on the Internet and it also reduces network traffic when communicating with the database
- VB .NET now supports Multithreading. A threaded application allows to do number of different things at once, running different execution threads allowing to use system resources
- Web Development is now an integral part of VB .NET making Web Forms and Web Services two major types of applications

Namespaces

A namespace is a collection of different classes. All VB applications are developed using classes from the .NET System namespace. The namespace with all the built-in VB functionality is the System namespace. All other namespaces are based on this System namespace.

Some Namespaces and their use:

System: Includes essential classes and base classes for commonly used data types, events, exceptions and so on
System.Collections: Includes classes and interfaces that define various collection of objects such as list, queues,
hash tables, arrays, etc
System.Data: Includes classes which lets us handle data from data sources
System.Data.OleDb: Includes classes that support the OLEDB .NET provider
System.Data.SqlClient: Includes classes that support the SQL Server .NET provider
System.Diagnostics: Includes classes that allow to debug our application and to step through our code
System.Drawing: Provides access to drawing methods
System.Globalization: Includes classes that specify culture-related information
System.IO: Includes classes for data access with Files
System.Net: Provides interface to protocols used on the internet
System.Reflection: Includes classes and interfaces that return information about types, methods and fields
System.Security: Includes classes to support the structure of common language runtime security system
System.Threading: Includes classes and interfaces to support multithreaded applications
System.Web: Includes classes and interfaces that support browser-server communication
System.Web.Services: Includes classes that let us build and use Web Services
System.Windows.Forms: Includes classes for creating Windows based forms
System.XML: Includes classes for XML support

Assemblies

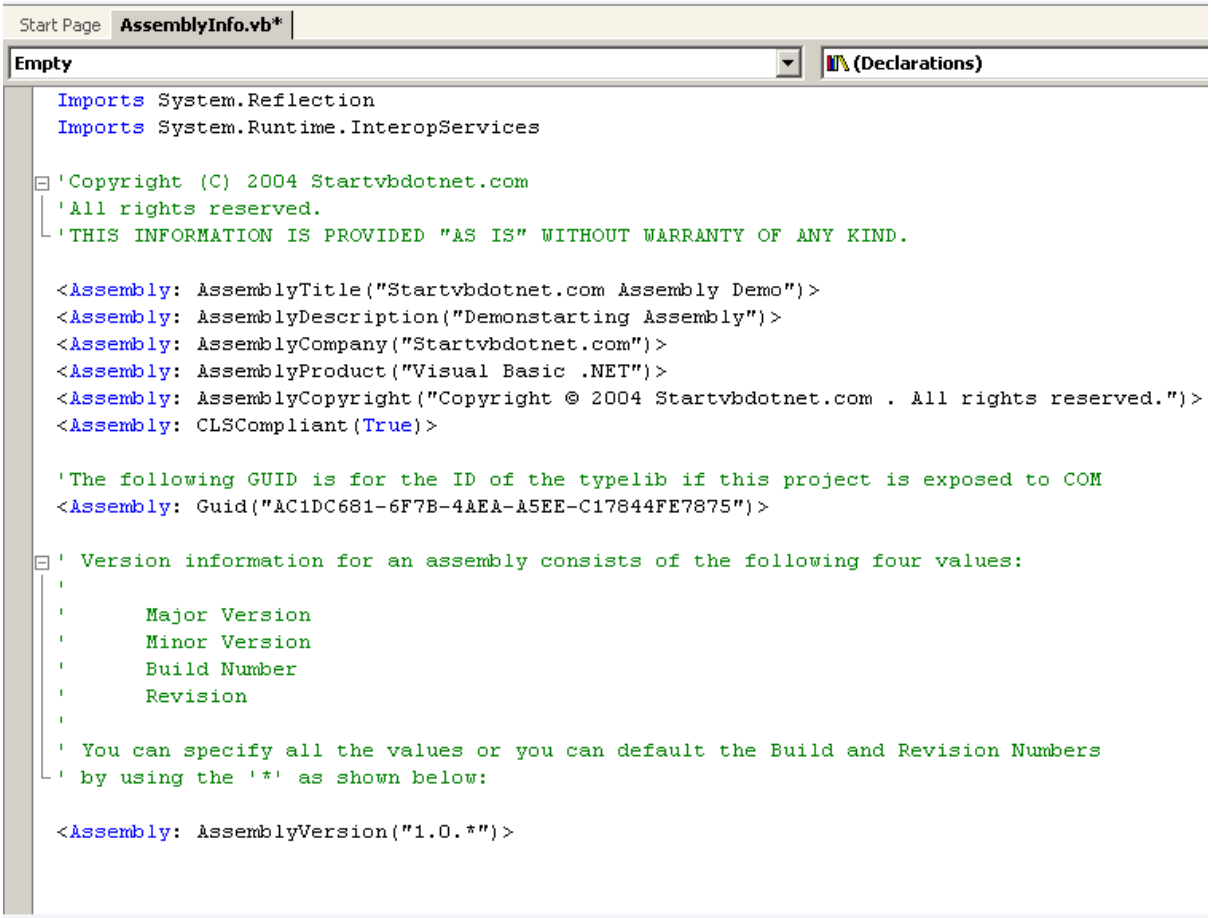
An assembly is the building block of a .NET application. It is a self describing collection of code, resources, and metadata (data about data, example, name, size, version of a file is metadata about that file). An Assembly is a complied and versioned collection of code and metadata that forms an atomic functional unit. Assemblies take the form of a dynamic link library (.dll) file or executable program file (.exe) but they differ as they contain the information found in a type library and the information about everything else needed to use anapplication or component. All .NET programs are constructed from these Assemblies. Assemblies are made of two parts: manifest, contains information about what is contained within the assembly and modules, internal files of IL code which are ready to run. When programming, we don't directly deal with assemblies as the CLR and the .NET framework takes care of that behind the scenes. The assembly file is visible in the Solution Explorer window of the project.

An assembly includes:

- Information for each public class or type used in the assembly – information includes class or type names, the classes from which an individual class is derived, etc
- Information on all public methods in each class, like, the method name and return values (if any)
- Information on every public parameter for each method like the parameter's name and type
- Information on public enumerations including names and values

- Information on the assembly version (each assembly has a specific version number)
- Intermediate language code to execute
- A list of types exposed by the assembly and list of other assemblies required by the assembly

Image of a Assembly file is displayed below.

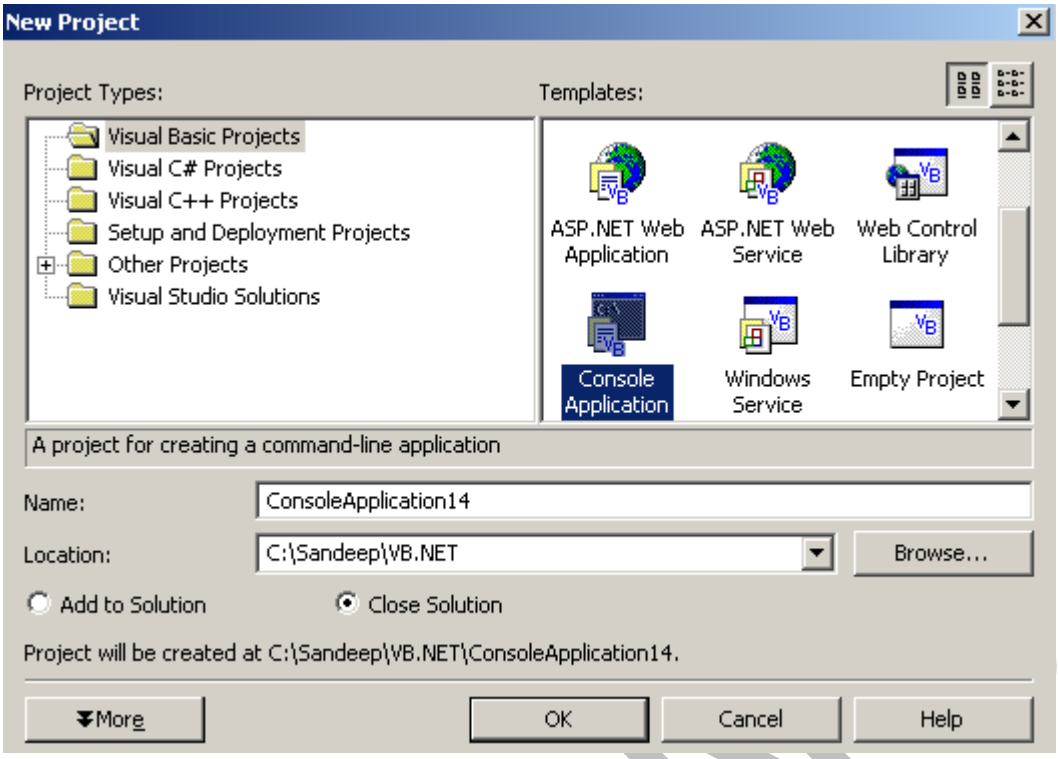


Console Applications

Console Applications are command-line oriented applications that allow us to read characters from the console, write characters to the console and are executed in the DOS version. Console Applications are written in code and are supported by the System.Console namespace.

Example on a Console Application

Create a folder in C: drive with any name (say, examples) and make sure the console applications which you open are saved there. That's for your convenience. The default location where all the .NET applications are saved is C:\Documents and Settings\Administrator\My Documents\Visual Studio Projects. The new project dialogue looks like the image below.

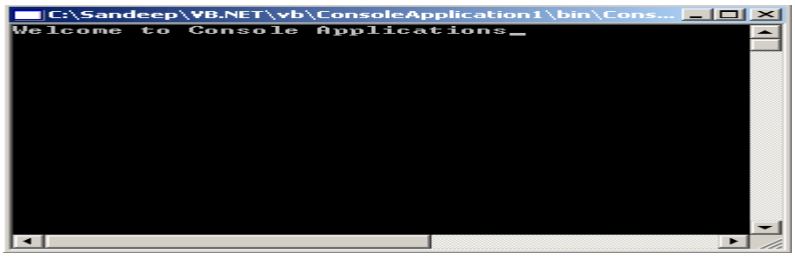


The following code is example of a console application:

```
Module Module1

Sub Main()
System.Console.Write("Welcome to Console Applications")
End Sub

End Module
```

You run the code by selecting Debug->Start from the main menu or by pressing F5 on the keyboard. The result "Welcome to Console Applications" displayed on a DOS window. Alternatively, you can run the program using the VB compiler (vbc). To do that, go to the Visual Studio. NET command prompt selecting from Start->Programs->Visual Studio.NET->Visual Studio.NET Tools->Visual Studio.NET Command Prompt and type:
c:\examples>vbc example1.vb.

The result, "Welcome to Console Applications" is displayed on a DOS window as shown in the image below.



Breaking the Code to understand it

Note the first line, we're creating a Visual Basic Module and Modules are designed to hold code. All the code which we write should be within the Module.
Next line starts with Sub Main(), the entry point of the program.
The third line indicates that we are using the Write method of the System.Console class to write to the console.

Commenting the Code

Comments in VB.NET begin with a single quote (') character and the statements following that are ignored by the compiler. Comments are generally used to specify what is going on in the program and also gives an idea about the flow of the program. The general form looks like this:

```
Dim I as Integer
'declaring an integer

Code
```

Data Types, Access Specifiers

Data Types in VB .NET

The Data types available in VB .NET, their size, type, description are summarized in the table below.

Data Type	Size in Bytes	Description	Type
Byte	1	8-bit unsigned integer	System.Byte
Char	2	16-bit Unicode characters	System.Char
Integer	4	32-bit signed integer	System.Int32
Double	8	64-bit floating point variable	System.Double
Long	8	64-bit signed integer	System.Int64
Short	2	16-bit signed integer	System.Int16
Single	4	32-bit floating point variable	System.Single
String	Varies	Non-Numeric Type	System.String
Date	8		System.Date
Boolean	2	Non-Numeric Type	System.Boolean

Object	4	Non-Numeric Type	System.Object
Decimal	16	128-bit floating point variable	System.Decimal

Access Specifiers

Access specifiers let's us specify how a variable, method or a class can be used. The following are the most commonly used one's:

Public: Gives variable public access which means that there is no restriction on their accessibility
Private: Gives variable private access which means that they are accessible only within their declaration content
Protected: Protected access gives a variable accessibility within their own class or a class derived from that class
Friend: Gives variable friend access which means that they are accessible within the program that contains their declaration
Protected Friend: Gives a variable both protected and friend access
Static: Makes a variable static which means that the variable will hold the value even the procedure in which they are declared ends
Shared: Declares a variable that can be shared across many instances and which is not associated with a specific instance of a class or structure
ReadOnly: Makes a variable only to be read and cannot be written

Variables

Variables are used to store data. A variable has a name to which we refer and the data type, the type of data the variable holds. VB .NET now needs variables to be declared before using them. Variables are declared with the Dim keyword. Dim stands for Dimension.

Example

```
Imports System.Console
Module Module1

Sub Main()
Dim a,b,c as Integer
'declaring three variables of type integer
a=10
b=20
c=a+b
Write("Sum of a and b is" & c)
End Sub

End Module
```

Statements and Scope

Statements

A statement is a complete instruction. It can contain keywords, operators, variables, literals, expressions and constants. Each statement in Visual Basic should be either a declarationstatement or a executable statement. A declaration statement is a statement that can create a variable, constant, data type. They are the one's we generally use to declare our variables. On the other hand, executable statements are the statements that perform an action. They execute a series of statements. They can execute a function, method, loop, etc.

Option Statement

The Option statement is used to set a number of options for the code to prevent syntax and logical errors. This statement is normally the first line of the code. The Option values inVisual Basic are as follows.

Option Compare: You can set it's value to Text or Binary. This specifies if the strings are compared using binary or text comparison operators.
Option Explicit: Default is On. You can set it to Off as well. This requires to declare all the variables before they are used.
Option Strict: Default is Off. You can set it to On as well. Used normally when working with conversions in code. If you want to assign a value of one type to another then you should set it to On and use the conversion functions else Visual Basic will consider that as an error.

Example of Option Statement

The following code demonstrates where to place the Option statement.

```
Option Strict Off
Imports System
Module Module 1

Sub Main ()
Console.WriteLine ("Using Option")
End Sub

End Module
```

The following code throws an error because Option Strict is On and the code attempts to convert a value of type double to integer.

```
Option Strict On
Imports System.Console
Module Module2

Sub Main()
Dim i As Integer
Dim d As Double = 20.12
```

```
i = d
WriteLine(i)
End Sub

End Module
```

We always should program with Option Strict On. Doing so allows us to catch many errors at compile time that would otherwise be difficult to track at run time.

Imports Statement

The Imports statement is used to import namespaces. Using this statement prevents you to list the entire namespace when you refer to them.

Example of Imports Statement

The following code imports the namespace System.Console and uses the methods of that namespace preventing us to refer to it every time we need a method of this namespace.

```
Imports System.Console
Module Module1

Sub Main()
Write("Imports Statement")
WriteLine("Using Import")
End Sub

End Module
```

The above two methods without an imports statement would look like this: System.Console.Write("Imports Statement") and System.Console.WriteLine("Using Import")

With Statement

With statemnt is used to execute statements using a particular object. The syntax looks like this:

```
With object
[statements]
End With
```

Sample Code

The following code sets text and width for a button using the With Statement.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
```

```
System.EventArgs)_
Handles MyBase.Load
With Button1
.Text = "With Statement"
.Width = 150
End With
End Sub
```

Boxing

Boxing is implicit conversion of value types to reference types. Recall that all classes and types are derived from the Object class. Because they are derived from Object class they can be implicitly converted to that type. The following sample shows that:

```
Dim x as Integer=20
'declaring an integer x
Dim o as Object
'declaring an object
o=x
converting integer to object
```

Unboxing is the conversion of a boxed value back to a value type.

Scope

The scope of an element in code is all the code that can refer to it without qualifying it's name. Stated other way, an element's scope is it's accessibility in code. Scope is normally used when writing large programs as large programs divide code into different classes, modules, etc. Also, scope prevents the chance of code referring to the wrong item. The different kinds of scope available in VB .NET are as follows:

Block Scope: The element declared is available only within the code block in which it is declared.
Procedure Scope: The element declared is available only within the procedure in which it is declared.
Module Scope: The element is available to all code within the module and class in which it is declared.
Namespace Scope: The element declared is available to all code

Methods

A Method is a procedure built into the class. They are a series of statements that are executed when called. Methods allow us to handle code in a simple and organized fashion. There are two types of methods in VB .NET: those that return a value (Functions) and those that do not return a value (Sub Procedures). Both of them are discussed below.

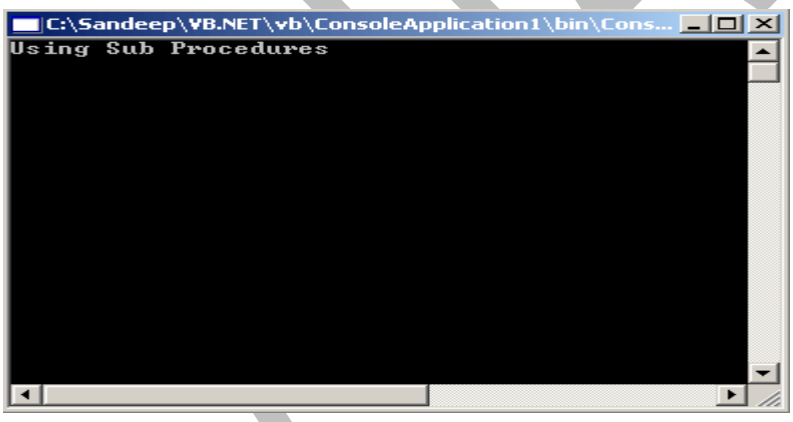
Sub Procedures

Sub procedures are methods which do not return a value. Each time when the Sub procedure is called the statements within it are executed until the matching End Sub is encountered. Sub Main(), the starting point of the program itself is a sub procedure. When the application starts execution, control is transferred to Main Sub procedure automatically which is called by default.

Example of a Sub Procedure

```
Module Module1

Sub Main()
'sub procedure Main() is called by default
Display()
'sub procedure display() which we are creating
End Sub

Sub Display()
System.Console.WriteLine("Using Sub Procedures")
'executing sub procedure Display()
End Sub
End Module
```

The image below displays output from above code.



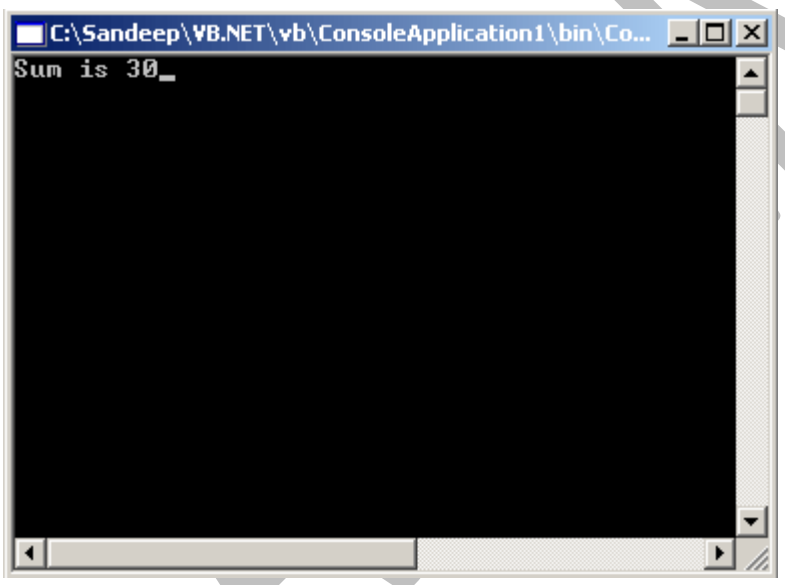
Functions

Function is a method which returns a value. Functions are used to evaluate data, make calculations or to transform data. Declaring a Function is similar to declaring a Sub procedure. Functions are declared with the Function keyword. The following code is an example on Functions:

```
Imports System.Console
Module Module1
```

```
Sub Main()
Write("Sum is" & " " & Add())
'calling the function
End Sub

Public Function Add() As Integer
'declaring a function add
Dim i, j As Integer
'declaring two integers and assigning values to them
i = 10
j = 20
Return (i + j)
'performing the sum of two integers and returning it's value
End Function

End Module
```

The image below displays output from above code.



Calling Methods

A method is not executed until it is called. A method is called by referencing it's name along with any required parameters. For example, the above code called the Add method in Sub main like this:
Write("Sum is" & " " & Add()).

Method Variables

Variables declared within methods are called method variables. They have method scope which means that once the method is executed they are destroyed and their memory is reclaimed. For example, from the above code (Functions) the Add method declared two

integer variables i, j. Those two variables are accessible only within the method and not from outside the method.

Parameters

A parameter is an argument that is passed to the method by the method that calls it. Parameters are enclosed in parentheses after the method name in the method declaration. You must specify types for these parameters. The general form of a method with parameters looks like this:

```
Public Function Add(ByVal x1 as Integer, ByVal y1 as Integer)
------------
Implementation
------------
End Function
```

Conditional Statements

If....Else statement

If conditional expression is one of the most useful control structures which allows us to execute a expression if a condition is true and execute a different expression if it is False. The syntax looks like this:

```
If condition Then
[statements]
Else If condition Then
[statements]
-
-
Else
[statements]
End If
```

Understanding the Syntax

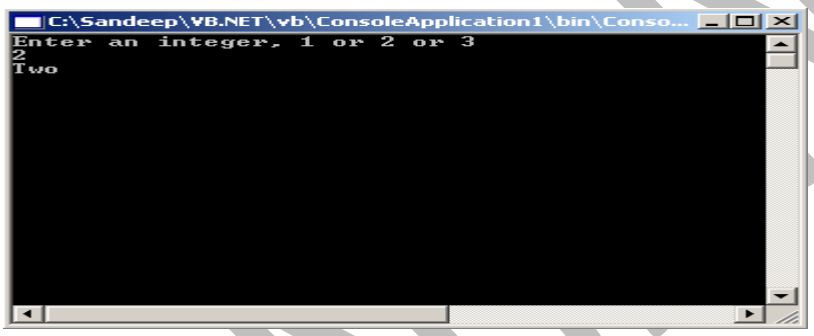
If the condition is true, the statements following the Then keyword will be executed, else, the statements following the ElseIf will be checked and if true, will be executed, else, the statements in the else part will be executed.

Example

```
Imports System.Console
Module Module1

Sub Main()
Dim i As Integer
WriteLine("Enter an integer, 1 or 2 or 3")
i = Val(ReadLine())
```

```
'ReadLine() method is used to read from console

If i = 1 Then
WriteLine("One")
ElseIf i = 2 Then
WriteLine("Two")
ElseIf i = 3 Then
WriteLine("Three")
Else
WriteLine("Number not 1,2,3")
End If

End Sub

End Module
```

The image below displays output from above code.



Select....Case Statement

The Select Case statement executes one of several groups of statements depending on the value of an expression. If your code has the capability to handle different values of a particular variable then you can use a Select Case statement. You use Select Case to test an expression, determine which of the given cases it matches and execute the code in that matched case.The syntax of the Select Case statement looks like this:

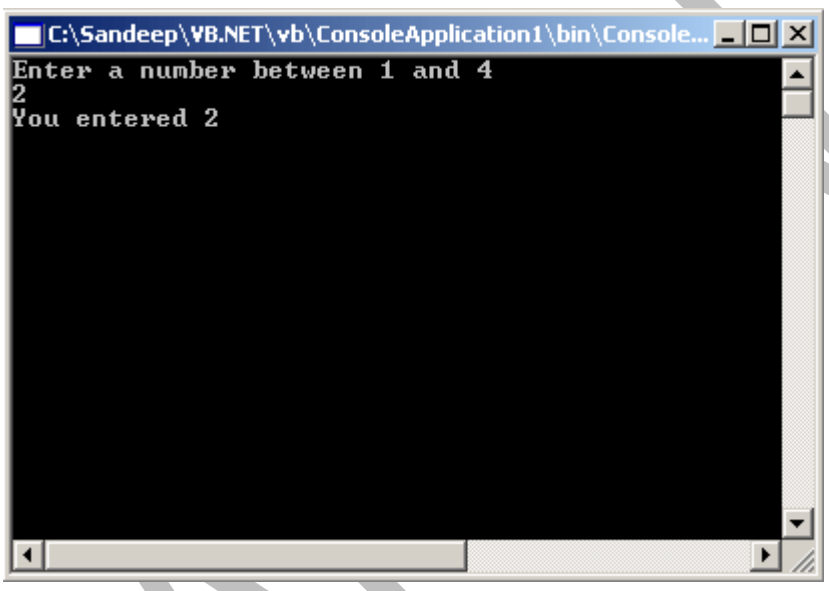
```
Select Case testexpression
   [Case expressionlist-n
      [statements-n]] . . .
   [Case Else elsestatements]
End Select
```

Example

```
Imports System.Console
Module Module1
Sub Main()
Dim keyIn As Integer
```

```
WriteLine("Enter a number between 1 and 4")
keyIn = Val(ReadLine())
Select Case keyIn
Case 1
WriteLine("You entered 1")
Case 2
WriteLine("You entered 2")
Case 3
WriteLine("You entered 3")
Case 4
WriteLine("You entered 4")
End Select
End Sub
End Module
```

The image below displays output from above code.



Loops

For Loop

The For loop is the most popular loop. For loops enable us to execute a series of expressions multiple numbers of times. The For loop in VB .NET needs a loop index whichcounts the number of loop iterations as the loop executes. The syntax for the For loop looks like this:

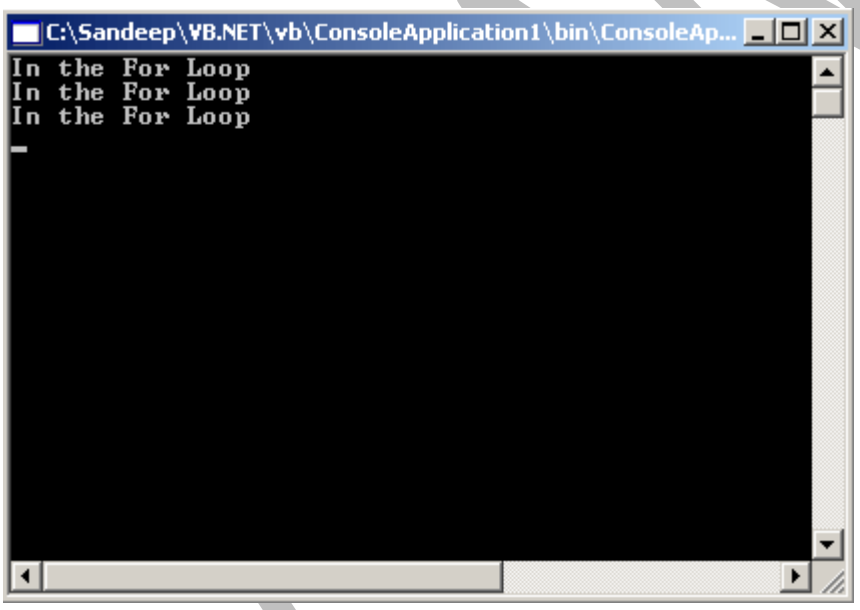
```
For index=start to end[Step step]
[statements]
[Exit For]
[statements]
Next[index]
```

The index variable is set to start automatically when the loop starts. Each time in the loop, index is incremented by step and when index equals end, the loop ends.

Example on For loop

```
Module Module1

Sub Main()
Dim d As Integer
For d = 0 To 2
System.Console.WriteLine("In the For Loop")
Next d
End Sub

End Module
```

The image below displays output from above code.



While loop

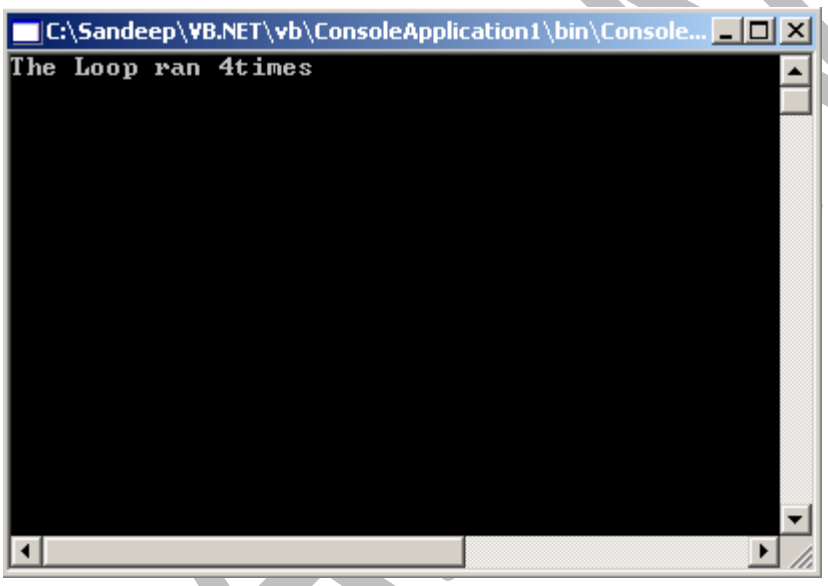
While loop keeps executing until the condition against which it tests remain true.
The syntax of while loop looks like this:

```
While condition
[statements]
End While
```

Example on While loop

```
Module Module1

Sub Main()
Dim d, e As Integer
d = 0
e = 6
While e > 4
e -= 1
d += 1
End While
System.Console.WriteLine("The Loop ran " & e & "times")
End Sub

End Module
```

The image below displays output from above code.



Do Loop

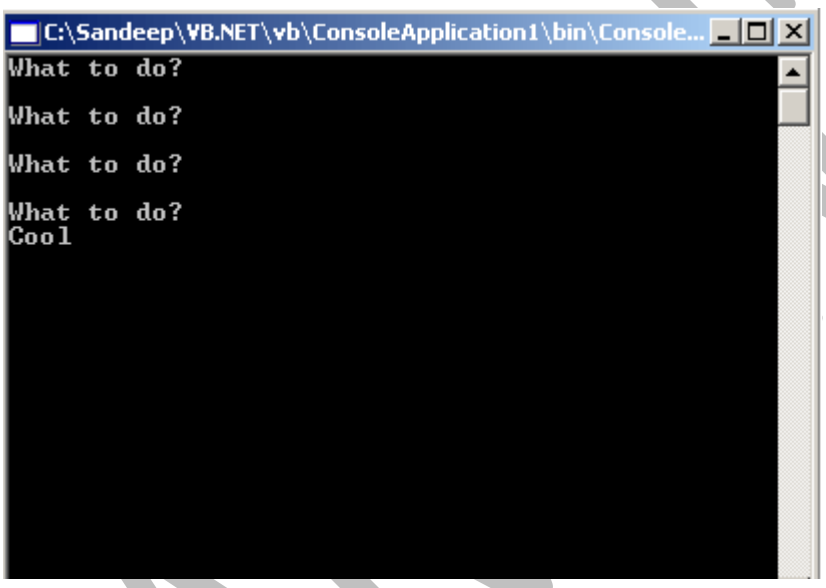
The Do loop can be used to execute a fixed block of statements indefinite number of times. The Do loop keeps executing it's statements while or until the condition is true. Two keywords, while and until can be used with the do loop. The Do loop also supports an Exit Do statement which makes the loop to exit at any moment. The syntax of Do loop looks like this:

Do[{while | Until} condition]
[statements]
[Exit Do]
[statements]
Loop

Example on Do loop

```
Module Module1

Sub Main()
Dim str As String
Do Until str = "Cool"
System.Console.WriteLine("What to do?")
str = System.Console.ReadLine()
Loop
End Sub

End Module
```

The image below displays output from above code.



Data Type Conversion, File Extensions

Converting between Data types

In Visual Basic, data can be converted in two ways: implicitly, which means the conversion is performed automatically, and explicitly, which means you must perform the conversion.

Implict Conversions

Let's understand implicit conversions in code. The example below declares two variable, one of type double and the other integer. The double data type is assigned a value and is converted to integer type. When you run the code the result displayed is an integer value, in this case the value displayed is 132 instead of 132.31223.

```
Imports System.Console
Module Module1
```

```
Sub Main()
Dim d=132.31223 as Double
Dim i as Integer
i=d
WriteLine("Integer value is" & i)
End Sub

End Module
```

Explicit Conversions

When types cannot be implicitly converted you should convert them explicitly. This conversion is also called as cast. Explicit conversions are accomplished using CTypefunction.

CType function for conversion

If we are not sure of the name of a particular conversion function then we can use the CType function. The above example with a CType function looks like this:

```
Imports System.Console
Module Module1

Sub Main()
Dim d As Double
d = 132.31223
Dim i As Integer
i = CType(d, i)
'two arguments, type we are converting from, to type desired
WriteLine("Integer value is" & i)
End Sub

End Module
```

Below is the list of conversion functions which we can use in VB .NET.

CBool - use this function to convert to Bool data type
CByte - use this function to convert to Byte data type
CChar - use this function to convert to Char data type
CDate - use this function to convert to Date type
CDbl - use this function to convert to Double data type
CDec - use this function to convert to Decimal data type
CInt - use this function to convert to Integer data type
CLng - use this function to convert to Long data type
CObj - use this function to convert to Object type
CShort - use this function to convert to Short data type
CSng - use this function to convert to Single data type
CString - use this function to convert to String data type

Attributes

Attributes are those that lets us specify information about the items we are using in VB .NET. Attributes are enclosed in angle brackets(< >) and are used when VB .NET needs to know more beyond the standard syntax.

File Extensions in VB .NET

The files and their extensions which are created as part of the Windows Application Project and their meaning are summarized below:

.vbproj->A Visual Basic project
Form1.vb->A form's code
AssemblyInfo.VB->Information about an assembly, includes version information
.vbproj.user->Stores project user options
.sln->Solution file which stores solution's configuration
.suo-> Stores Solution user options
Form1.resx.NET->XML based resource template
bin directory->Directory for binary executables
obj directory->Directory for debugging binaries

Language Terminology

Briefly on some terminology when working with the language:

Module: Used to hold code
Variable: A named memory location of a specific data type used to hold some value
Procedure: A callable series of statements which may or may not return a value
Sub-Procedure: A procedure with no return value
Function: A procedure with return value
Methods: A procedure built into the class
Constructor: Special method used to initialize and customize the object. It has the same name as the class
Class: An OOP class which contains data and code
Object: An instance of a class
Arrays: Programming constructs that lets us access data by numeric index
Attributes: They are the items that specify information about other items being used in VB. NET

Operators

Visual Basic comes with many built-in operators that allow us to manipulate data. An operator performs a function on one or more operands. For example, we add two variables with the "+" addition operator and store the result in a third variable with the "=" assignment operator like this: int x + int y = int z. The two variables (x ,y) are called operands. There are different types of operators in Visual Basic and they are described below in the order of their precedence.

Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations that involve calculation of numeric values. The table below summarizes them:

Operator	Use
^	Exponentiation
-	Negation (used to reverse the sign of the given value, exp -intValue)
*	Multiplication
/	Division
\	Integer Division
Mod	Modulus Arithmetic
+	Addition
-	Subtraction

Concatenation Operators:- Concatenation operators join multiple strings into a single string. There are two concatenation operators, + and & as summarized below:

Operator	Use
+	**String Concatenation**
&	**String Concatenation**

Comparison Operators

A comparison operator compares operands and returns a logical value based on whether the comparison is true or not. The table below summarizes them:

Operator	Use
=	Equality
<>	Inequality
<	Less than
>	Greater than
>=	Greater than or equal to
<=	Less than or equal to

Logical / Bitwise Operators

The logical operators compare Boolean expressions and return a Boolean result. In short, logical operators are expressions which return a true or false result over a conditional expression. The table below summarizes them:

Operator	Use
Not	Negation
And	Conjunction
AndAlso	Conjunction
Or	Disjunction
OrElse	Disjunction
Xor	Disjunction

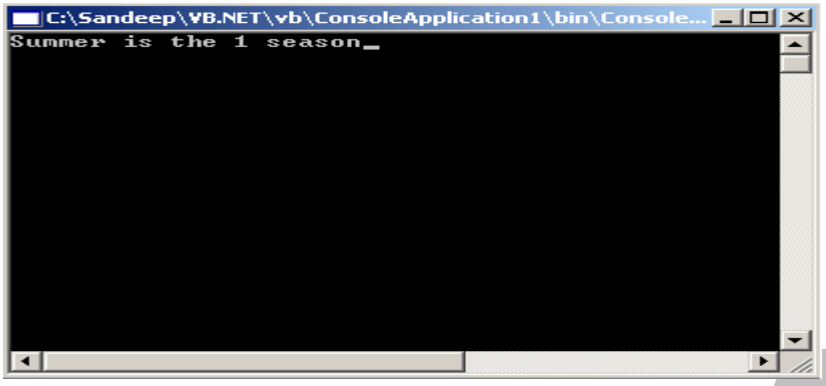
Enumeration

Enumeration is a related set of constants. They are used when working with many constants of the same type. It's declared with the Enum keyword.

Example

```
Imports System.Console
Module Module1

Enum Seasons
Summer = 1
Winter = 2
Spring = 3
Autumn = 4
End Enum

Sub Main()
Write("Summer is the" & Seasons.Summer & "season")
End Sub

End Module
```

Output of above code is the image below. To use a constant from the enumeration it should be referred like this, Seasons.Winter and so on.



Constants

When we have certain values that we frequently use while programming, we should use Constants. A value declared as constant is of fixed value that cannot be changed once set. Constants should be declared as Public if we want it to be accessed by all parts of the application. In Visual Basic .NET we use the Const keyword to declare a constant. The following line of code declares a constant: **Public Const Pi as Double=3.14159265**

Exception Handling

Exceptions are runtime errors that occur when a program is running and causes the program to abort without execution. Such kind of situations can be handled using Exception Handling. By placing specific lines of code in the application we can handle most of the errors that we may encounter and we can enable the application to continue running. VB .NET supports two ways to handle exceptions, Unstructured exception Handling using the on error goto statement and Structured exception handling usingTry....Catch.....Finally

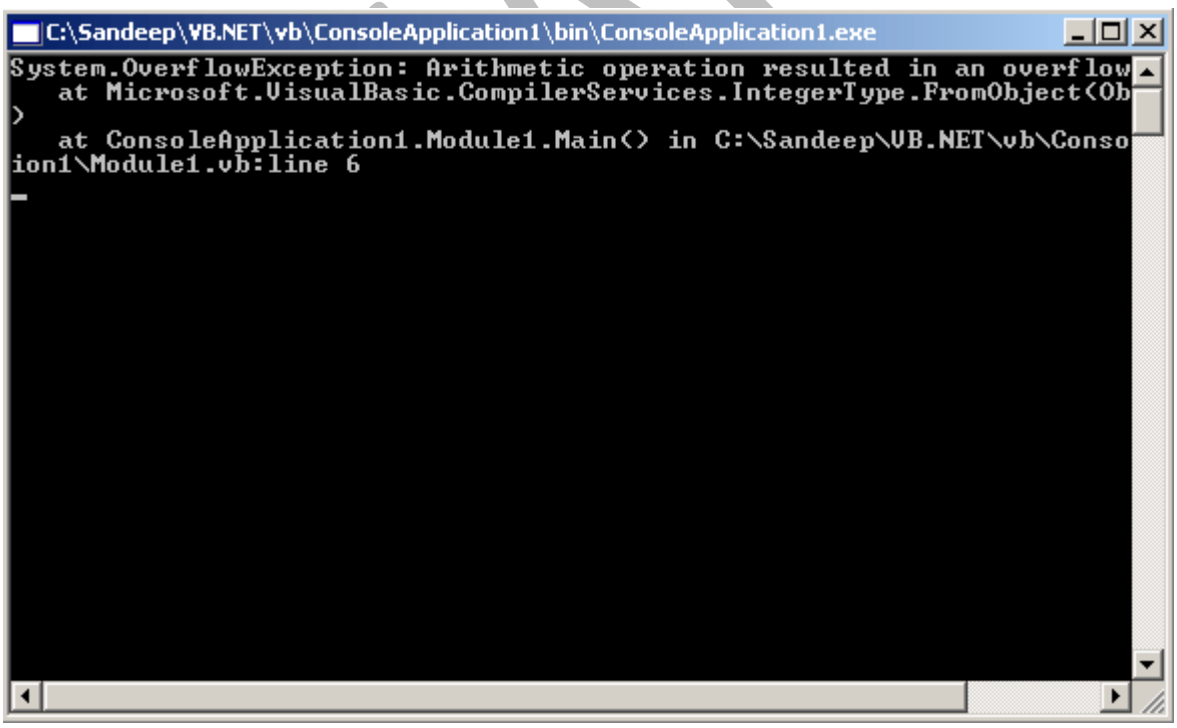
Let's look at the new kind of exception handling introduced in VB .NET which is the Structured Exception Handling. VB .NET uses Try....Catch....Finally block type exception handling. The syntax looks like this:

```
Module Module1
Sub Main()
Try
-
-
Catch e as Exception
-
-
Finally
End Try
End Sub
End Module
```

Example

```
Imports System.Console
Module Module1

Sub Main()
Dim a = 0, b = 1, c As Integer
Try
c = b / a
'the above line throws an exception
WriteLine("C is " & c)
Catch e As Exception
WriteLine(e)
'catching the exception
End Try
End Sub

End Module
```

The output of the above code displays a message stating the exception. The reason for the exception is because any number divided by zero is infinity. When working with Structured exception handling you can have multiple Catch blocks to handle different types of exceptions differently. The code in the Finally block is optional. If there is a Finally block in the code then that code is executed last.



Arrays

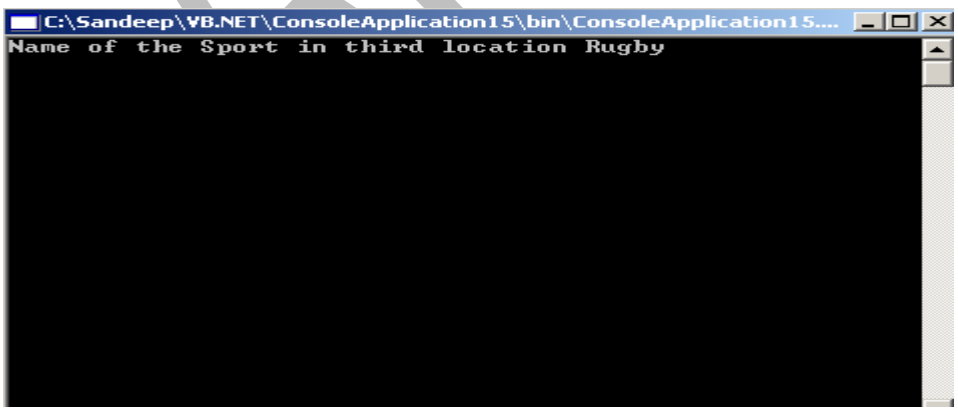
Arrays are programming constructs that store data and allow us to access them by numeric index or subscript. Arrays helps us create shorter and simpler code in many situations. Arrays in Visual Basic .NET inherit from the Array class in the System namespace. All arrays in VB are zero based, meaning, the index of the first element is zero

and they are numbered sequentially. You must specify the number of array elements by indicating the upper bound of the array. The upper bound is the numder that specifies the index of the last element of the array. Arrays are declared using Dim, ReDim, Static, Private, Public and Protected keywords. An array can have one dimension (liinear arrays) or more than one (multidimensional arrays). The dimensionality of an array refers to the number of subscripts used to identify an individual element. InVisual Basic we can specify up to 32 dimensions. Arrays do not have fixed size in Visual Basic.The following code demonstrates arrays.

```
Imports System.Console
Module Module1

Sub Main()
Dim sport(5) As String
'declaring an array
sport(0) = "Soccer"
sport(1) = "Cricket"
sport(2) = "Rugby"
sport(3) = "Aussie Rules"
sport(4) = "BasketBall"
sport(5) = "Hockey"
'storing values in the array
WriteLine("Name of the Sport in the third location" & " " & sport(2))
'displaying value from array
End Sub

End Module
```

Understanding the Code

The above code declared a sport array of type string like this: **Dim sport(5) as String**. This sport array has 6 elements starting from sport(0) to sport(5). The first element of an array is always referred by zero index. The image below displays output from above code.



You can also declare an array without specifying the number of elements on one line, you must provide values for each element when initializing the array. The following lines demonstrate that:

```
Dim Test() as Integer
'declaring a Test array
Test=New Integer(){1,3,5,7,9,}
```

Reinitializing Arrays

We can change the size of an array after creating them. The ReDim statement assigns a completely new array object to the specified array variable. You use ReDim statement to change the number of elements in an array. The following lines of code demonstrate that. This code reinitializes the Test array declared above.

```
Dim Test(10) as Integer
ReDim Test(25) as Integer
'Reinitializing the array
```

When using the Redim statement all the data contained in the array is lost. If you want to preserve existing data when reinitializing an array then you should use thePreserve keyword which looks like this:

```
Dim Test() as Integer={1,3,5}
'declares an array an initializes it with three members
ReDim Preserve Test(25)
'resizes the array and retains the the data in elements 0 to 2
```

Multidimensional Arrays

All arrays which were mentioned above are one dimensional or linear arrays. There are two kinds of multidimensional arrays supported by the .NET framework: Rectangulararrays and Jagged arrays.

Rectangular arrays

Rectangular arrays are arrays in which each member of each dimension is extended in each other dimension by the same length. We declare a rectangular array by specifying additional dimensions at declaration. The following lines of code demonstrate the declaration of a multidimensional array.

```
Dim rectArray(4, 2) As Integer
'declares an array of 5 by 3 members which is a 15 member array
Dim rectArray(,) As Integer = {{1, 2, 3}, {12, 13, 14}, {11, 10, 9}}
'setting initial values
```

Jagged Arrays

Another type of multidimensional array, Jagged Array, is an array of arrays in which the length of each array can differ. Example where this array can be used is to create a table in which the number of columns differ in each row. Say, if row1 has 3 columns, row2 has 3

columns then row3 can have 4 columns, row4 can have 5 columns and so on. The following code demonstrates jagged arrays.

```
Dim colors(2)() as String
'declaring an array of 3 arrays
colors(0)=New String(){"Red","blue","Green"}
initializing the first array to 3 members and setting values
colors(1)=New String(){"Yellow","Purple","Green","Violet"}
initializing the second array to 4 members and setting values
colors(2)=New String(){"Red","Black","White","Grey","Aqua"}
initializing the third array to 5 members and setting values
```

Strings, Math Functions

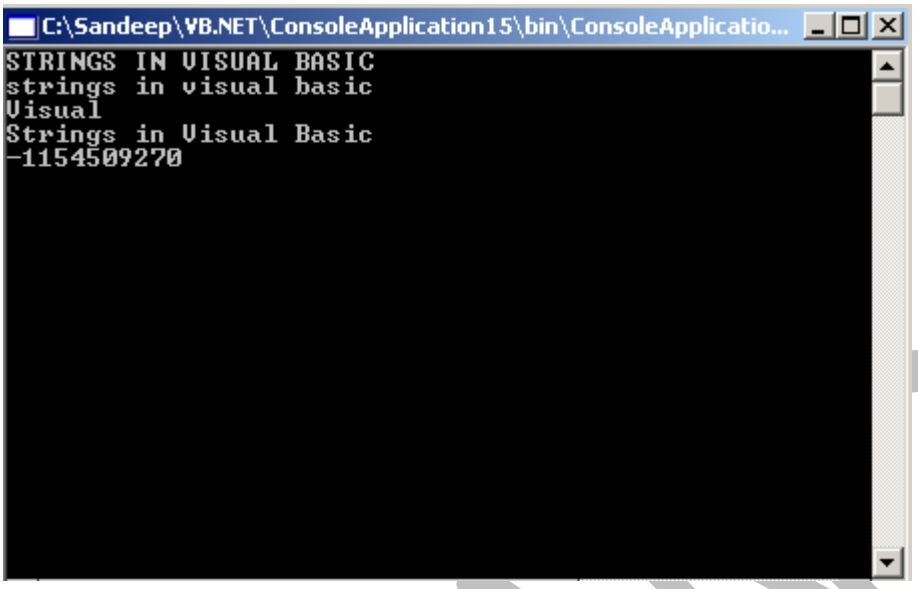
Strings

Strings in Visual Basic are supported by the .NET String class. The String data type can represent a series of characters and can contain approximately up to 2 billion Unicode characters. There are many built-in functions in the String class. Some .NET Framework functions are also built into the String class. The following code puts some Stringfunctions to work.

```
Imports System.Console
Module Module1

Sub Main()
Dim string1 As String = "Strings in Visual Basic"
Dim string2 As String
Dim string3 As String
Dim string4 As String
Dim string5 As String
Dim string6 As String
string2 = UCase(string1)
'converts string to uppercase
string3 = LCase(string1)
'converts string to lowercase
string4 = string1.Substring(11, 6)
'returns a substring
string5 = string1.Clone
'clones a string
string6 = string1.GetHashCode
'gets the hashcode
WriteLine(string2)
WriteLine(string3)
WriteLine(string4)
WriteLine(string5)
WriteLine(string6)
Read()
End Sub
```

```
End Module
```

The image below displays output from above code.



Math Functions

Visual Basic provides support for handling Mathematical calculations. Math functions are stored in System.Math namespace. We need to import this namespace when we work with Math functions. The functions built into Math class helps us calculate the Trigonometry values, Square roots, logarithm values, etc. The following code puts someMath functions to work.

```
Imports System.Console
Imports System.Math
Module Module1

Sub Main()
WriteLine("Sine 90 is" & " " & Sin(90))
'display Sine90 value
WriteLine("Square root of 81 is " & " " & Sqrt(81))
'displays square root of 81
WriteLine("Log value of 12 is" & " " & Log(12))
'displays the logarithm value of 12
Read()
End Sub

End Module
```

The image below displays output from above code.

Visual Studio .NET IDE

Visual Studio .NET IDE (Integrated Development Environment) is the Development Environment for all .NET based applications which comes with rich features. VS .NET IDE provides many options and is packed with many features that simplify application development by handling the complexities. Visual Studio .NET IDE is an enhancement to all previous IDE's by Microsoft.

Important Features

One IDE for all .NET Projects

Visual Studio .NET IDE provides a single environment for developing all types of .NET applications. Application's range from single windows applications to complex n-tierapplications and rich web applications.

Option to choose from Multiple Programming Languages

You can choose the programming language of your choice to develop applications based on your expertise in that language. You can also incorporate multiple programming languages in one .NET solution and edit that with the IDE.

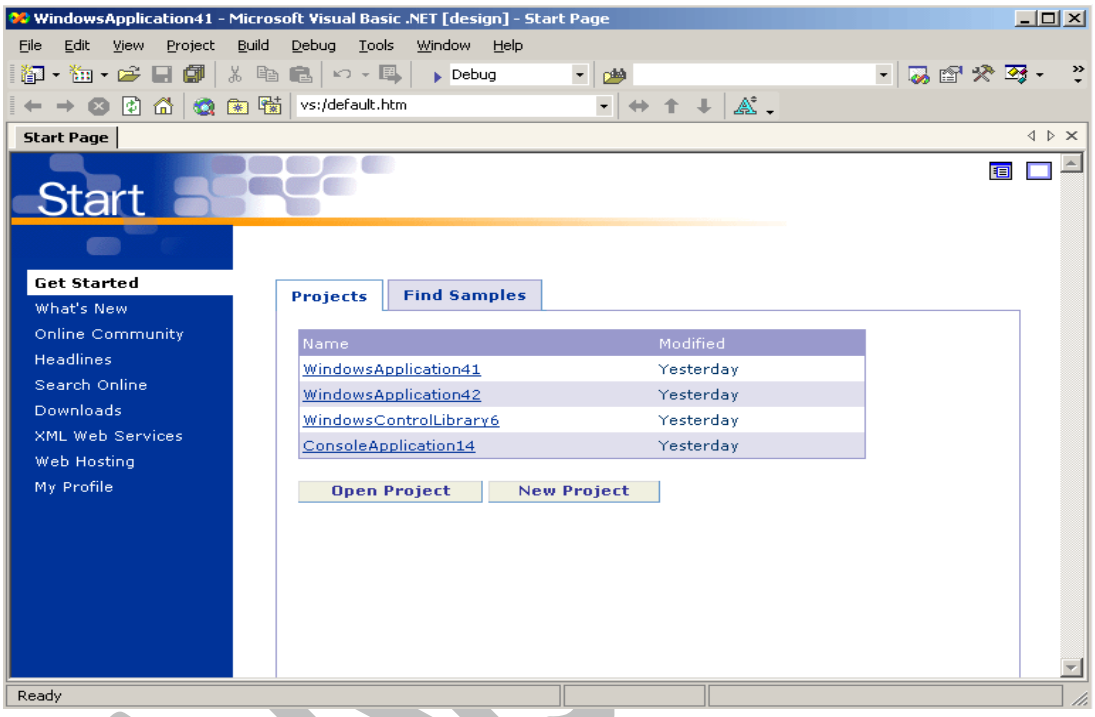
IDE is Customizable

You can customize the IDE based on your preferences. The My Profile settings allow you to do this. With these settings you can set the IDE screen the way you want, the way the keyboard behaves and you can also filter the help files based on the language of your choice.

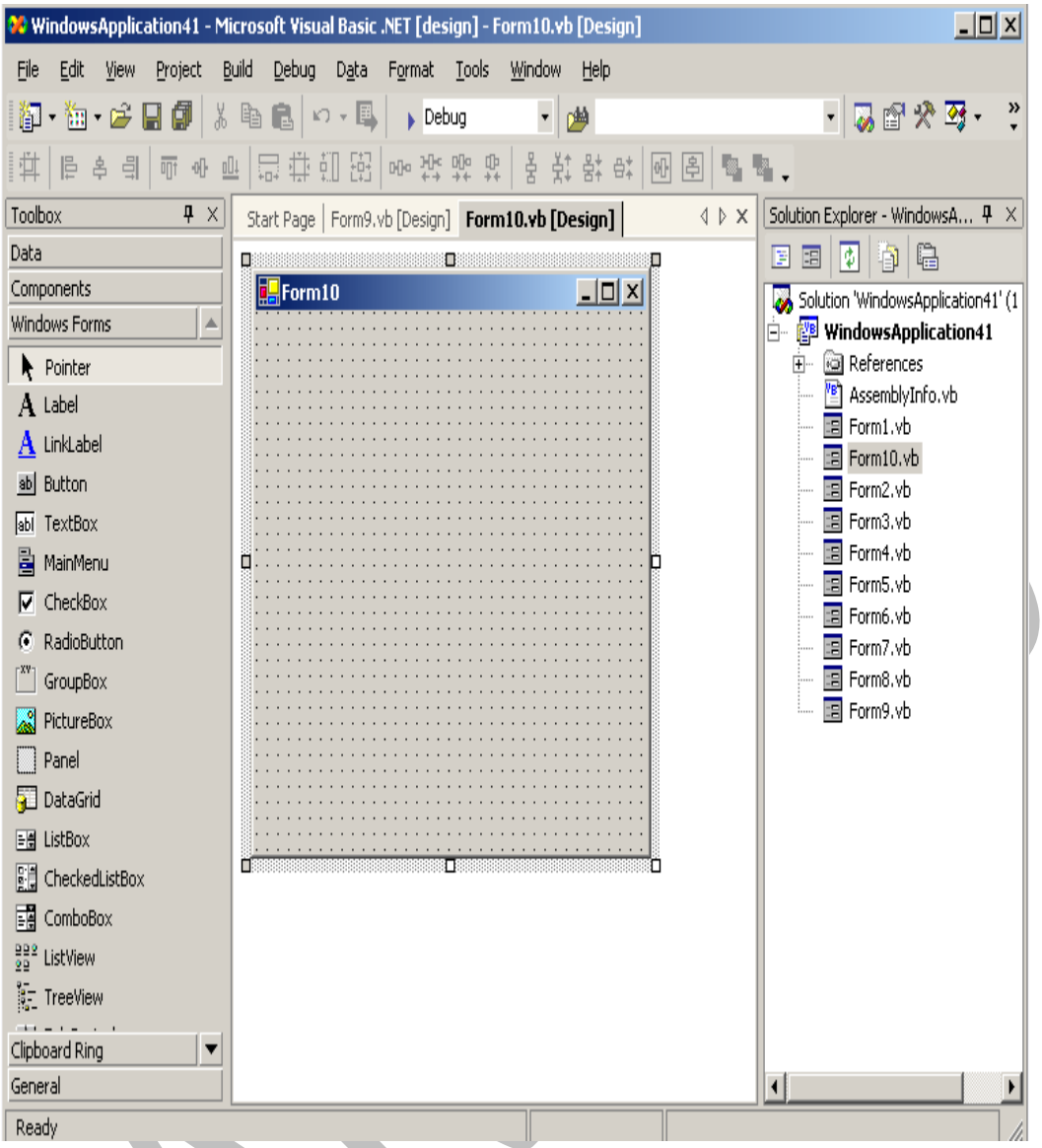
Built-in Browser

The IDE comes with a built-in browser that helps you browse the Internet without launching another application. You can look for additional resources, online help files, source codes and much more with this built-in browser feature.

When we open VS .NET from Start->Programs->Microsoft Visual Studio .NET->Microsoft Visual Studio .NET the window that is displayed first is the Start Page which is shown below. The start Page allows us to select from the most recent projects (last four projects) with which we worked or it can be customized based on your preferences.

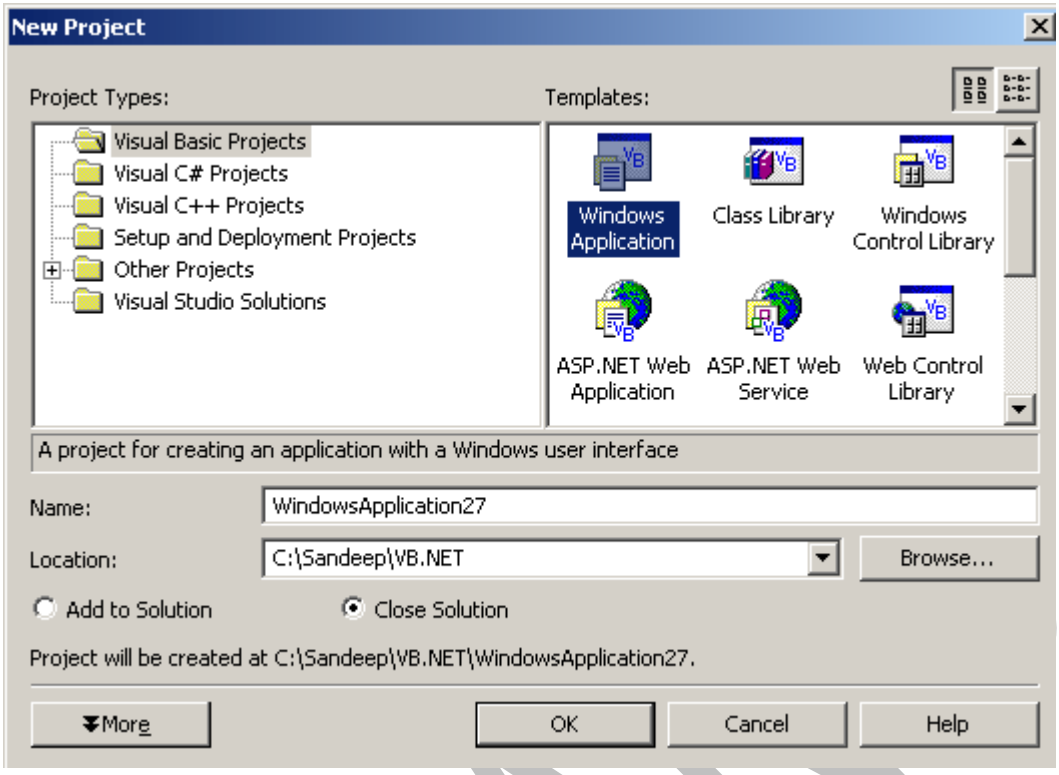


The Integrated Development Environment (IDE) shown in the image below is what we actually work with. This IDE is shared by all programming languages in Visual Studio. You can view the toolbars towards the left side of the image along with the Solution Explorer window towards the right.



New Project Dialogue Box

The New Project dialogue box like the one in the image below is used to create a new project specifying it's type allowing us to name the project and also specify it's location on the disk where it is saved. The default location on the hard disk where all the projects are saved isC:\DocumentsandSettings\Administrator\MyDocuments\VisualStudioProjects.



Following are different templates under Project Types and their use.

Windows Application: This template allows to create standard windows based applications.

Class Library: Class libraries are those that provide functionality similar to Active X and DLL by creating classes that access other applications.

Windows Control Library: This allows to create our own windows controls. Also called as User Controls, where you group some controls, add it to the toolbox and make it available to other projects.

ASP .NET Web Application: This allows to create web-based applications using IIS. We can create web pages, rich web applications and web services.

ASP .NET Web Service: Allows to create XML Web Services.

Web Control Library: Allows to create User-defined controls for the Web. Similar to user defined windows controls but these are used for Web.

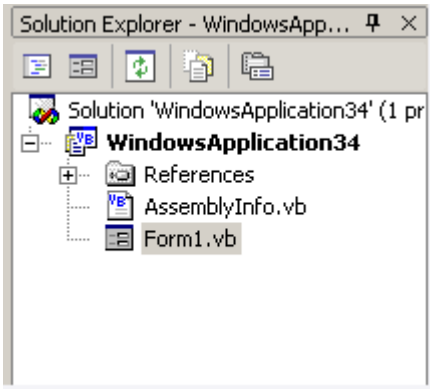
Console Application: A new kind of application in Visual Studio .NET. They are command line based applications.

Windows Service: These run continuously regardless of the user interaction. They are designed for special purpose and once written, will keep running and come to an end only when the system is shut down.

Other: This template is to develop other kinds of applications like enterprise applications, database applications etc.

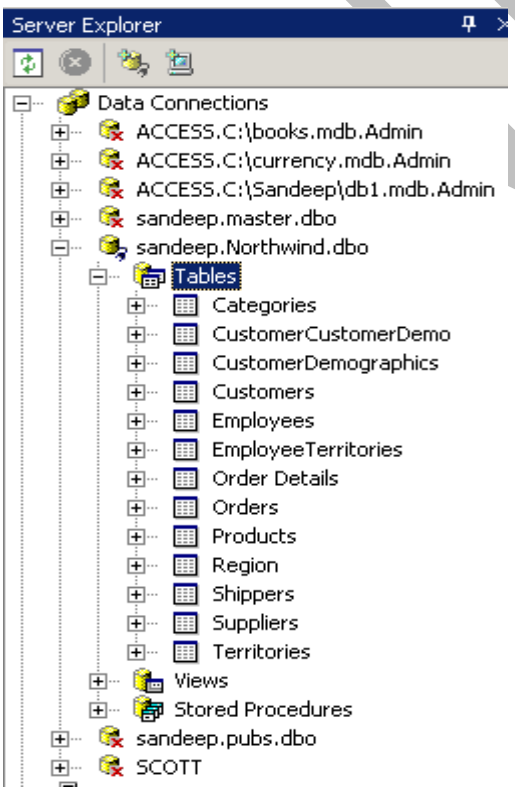
Solution Explorer Window

The Solution Explorer window gives an overview of the solution we are working with and lists all the files in the project. An image of the Solution Explorer window is shown below.



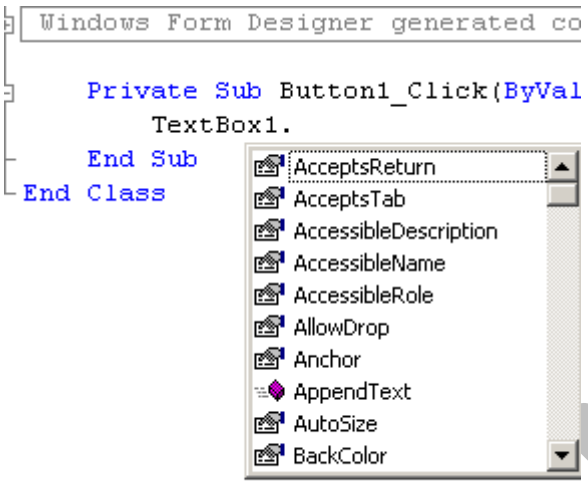
Server Explorer Window

The Server Explorer window is a great tool that provides "drag and drop" feature and helps us work with databases in an easy graphical environment. For example, if we drag and drop a database table onto a form, VB .NET automatically creates connection and command objects that are needed to access that table. The image below displays Server Explorer window.



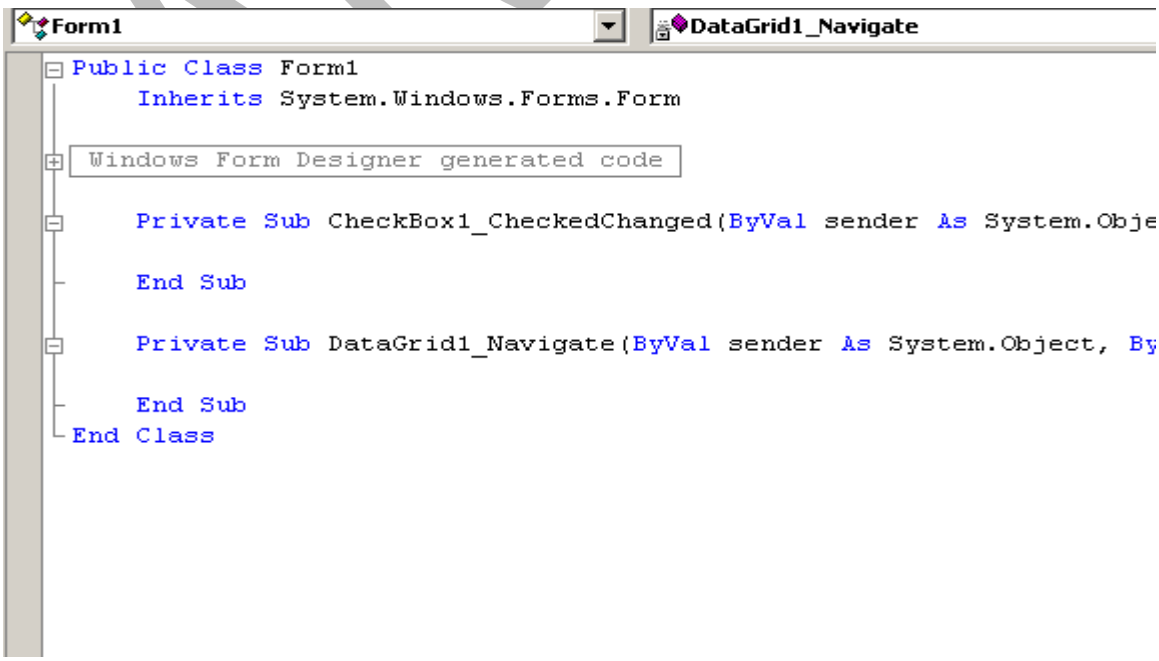
Intellisense

Intellisense is what that is responsible for the boxes that open as we type the code. IntelliSense provides a list of options that make language references easily accessible and helps us to find the information we need. They also complete the typing for us. The image below displays that.



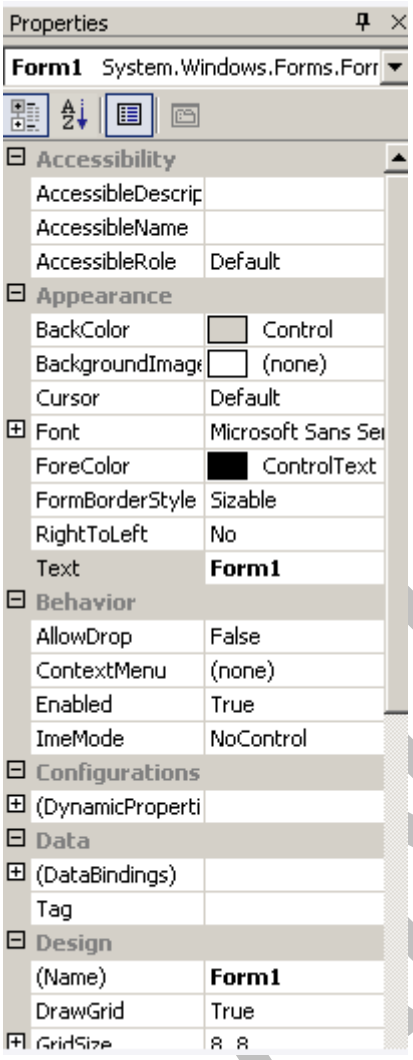
Code Designer Window

Code Designers like the image below allows us to edit and write code. This is the window that opens when we double-click on a form or any control. This is the place where we write all the code for the application. Notice the two drop-down list boxes at the top of the code window in the image below. The left box allows us to select the object's code we are working with and the right box allows us to select the part of code that we want to work. Also notice the "+" and "-" boxes in the code designer. You can use those boxes to display code Visual Basic .NET already created, like, Windows Forms Designer generated code, etc.



Properties Window

The properties window allows us to set properties for various objects at design time. For example, if you want to change the font, font size, backcolor, name, text that appears on a button, textbox etc, you can do that in this window. Below is the image of properties window. You can view the properties window by selecting
View->Properties Window from the main menu or by pressing F4 on the keyboard.

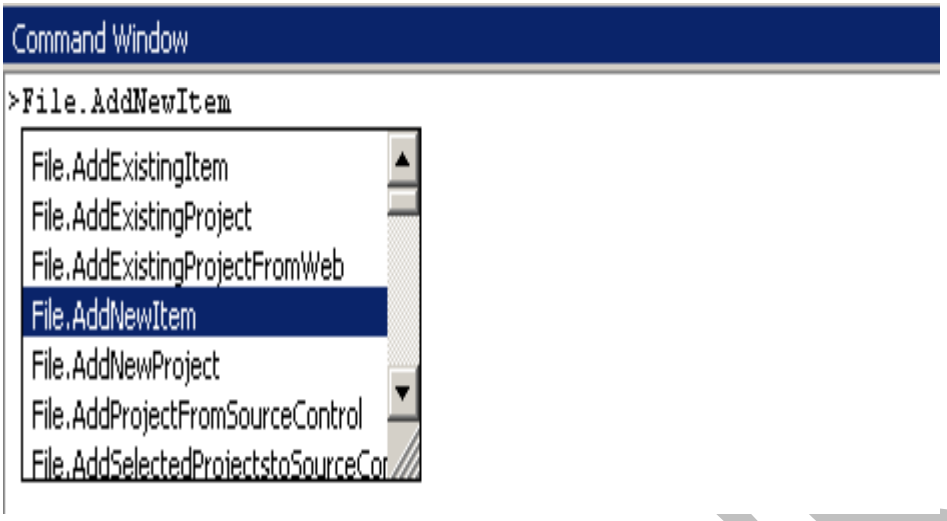


Dynamic Help Window

The dynamic help window displays help which looks up for things automatically. For example, if you want to get help with a form, select the form and select Help->Dynamic Help from the main menu. Doing that displays all the information relating to forms. The image below displays that. You can get help relating to anything with this feature. Say, if you want to know more about the form, select the form and select Dynamic Help from the Help menu. Doing that displays information about the form as shown in the image below..

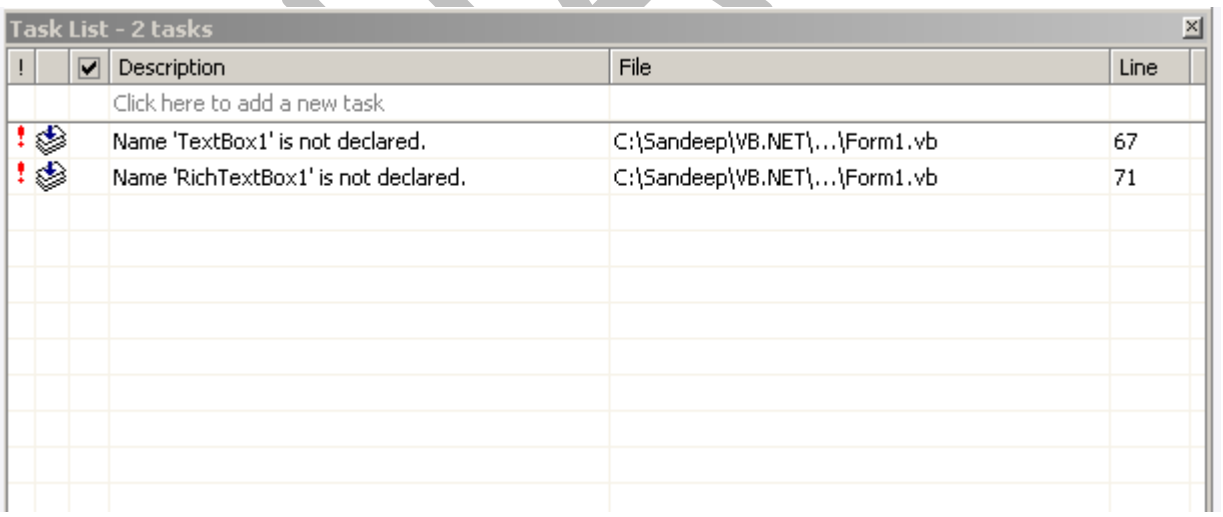
Command Window

The command window in the image below is a useful window. Using this window we can add new item to the project, add new project and so on. You can view the command window by selecting
View->Other Windows->Command Window from the main menu. The command window in the image displays all possible commands with File.



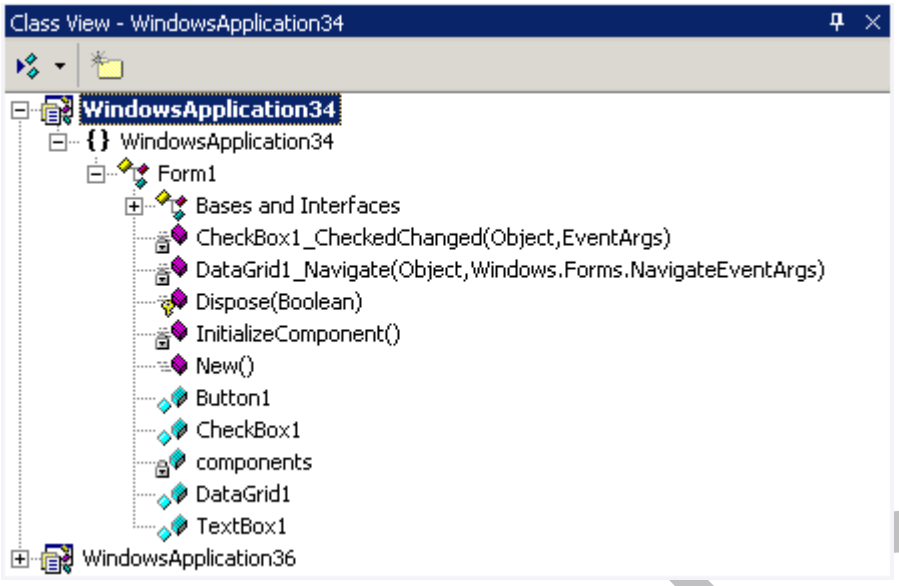
Task List Window

The task list window displays all the tasks that VB .NET assumes we still have to finish. You can view the task list window by selecting View->Show tasks->All or View->Other Windows->Task List from the main menu. The image below shows that. As you can see from the image, the task list displayed "TextBox1 not declared", "RichTextBox1 not declared". The reason for that message is, there were no controls on the form and attempts where made to write code for a textbox and a richtextbox. Task list also displays syntax errors and other errors you normally encounter during coding



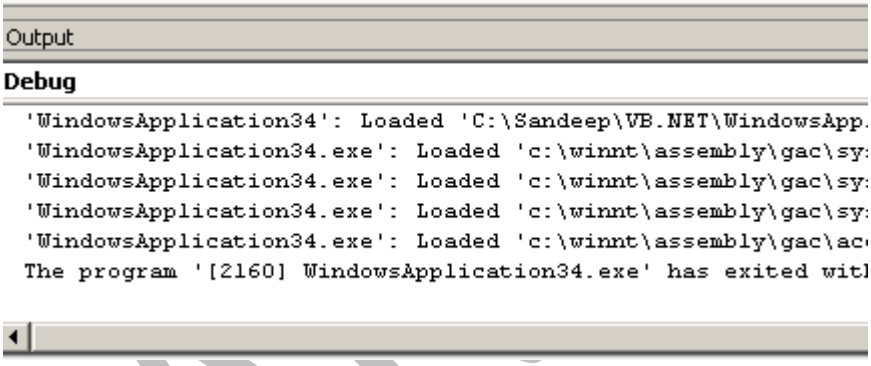
Class View Window

The class view window like the image below is the window that presents solutions and projects in terms of the classes they contain and the members of these classes. Using the class view window also helps us to find a member of a class that we want to work with. As you can notice from the image, the class view window displayed all the methods and events for the controls which were available on the form.



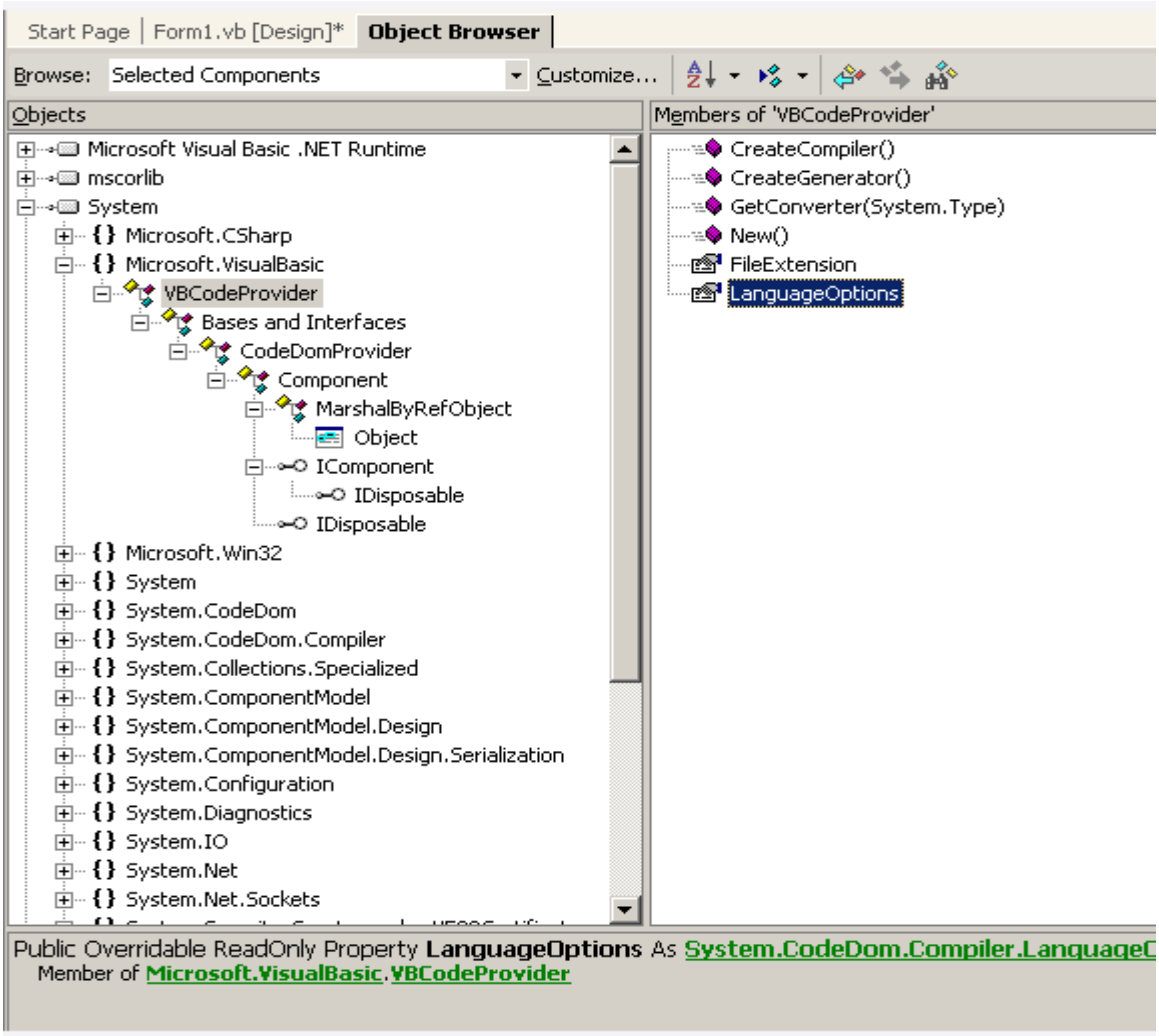
Output Window

The output window as you can see in the image below displays the results of building and running applications.



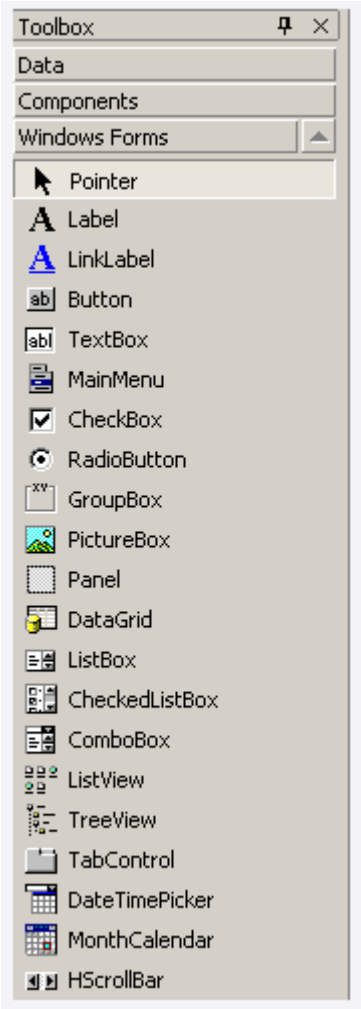
Object Explorer Window

The object explorer window allows us to view all the members of an object at once. It lists all the objects in our code and gives us access to them. The image below displays an object explorer window. You can view the object explorer window by selecting View->Other Windows-> Object Browser from the main menu.



Toolbox Window

The toolbox window is the window that gives us access to all controls, components, etc. As you can see from the image below, the toolbox uses tabs to divide it's contents into categories (Data, Components, Windows Forms and General). The Data tab displays tools for creating datasets and making data connections, the Windows Forms tab displays tools for adding controls to forms, the General tab is left empty by default, the Clipboard Ring tab displays recent items stored in the clipboard and allows us to select from them.



Shortcut Keys

Key	What it Does?
Ctrl + N	Opens the New Project Dialogue Box
Ctrl + Shift + O	Opens the Open File Dialog Box
Ctrl + Shift + A	Opens Add New Item window
Ctrl + D	Opens Add Existing Item window
Ctrl + S	Saves Current Form
Ctrl + Shift + S	Saves everything from Application
Alt + Q	Exits Visual Studio. NET
Ctrl + Z	Undo
Ctrl + Shift + Z	Redo
Ctrl + X	Cuts your selection

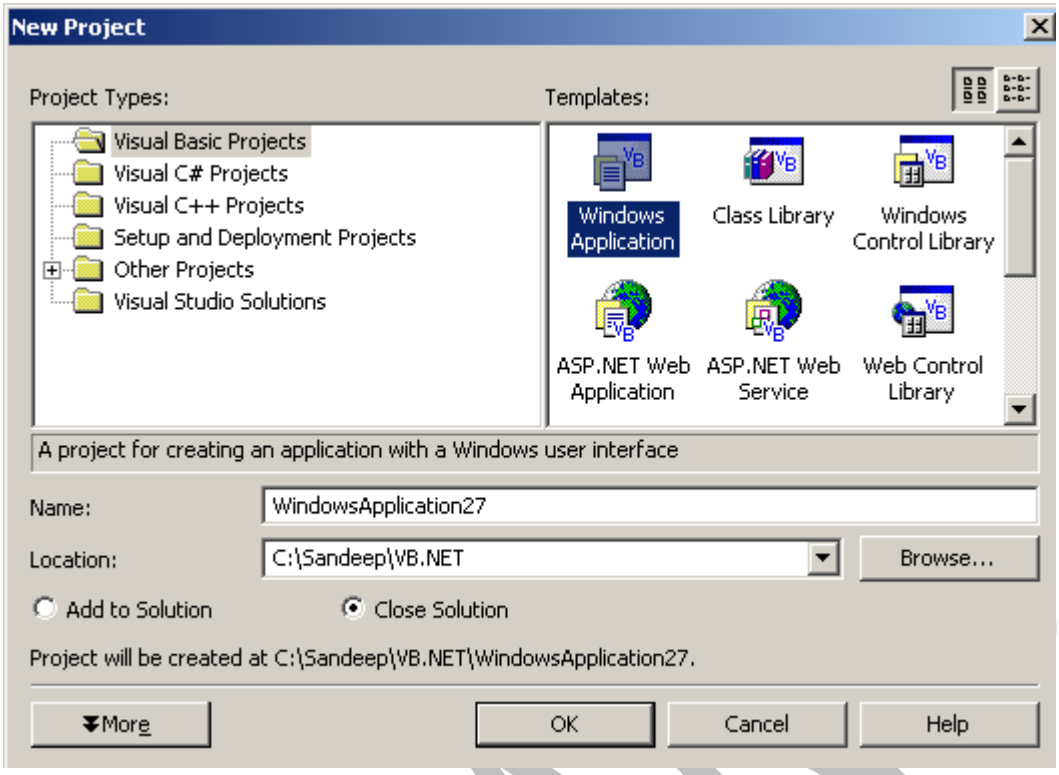
Ctrl + C	Copies your selection
Ctrl + V	Pastes your selection
Ctrl + A	Selects All
Del	Deletes your selection
Ctrl + F	Opens Find window
Ctrl + H	Opens Find and Replace window
Ctrl + Shift + H	Opens Replace in Files window
Ctrl + Alt + Shift + F12	Opens Find Symbol window
F7	Opens Code Designer window
Shift + F7	Gets you back to Design View
Ctrl + R	Opens the Solution Explorer window
Ctrl + Alt + S	Opens the Server Explorer window
Ctrl + Shift + C	Opens the Class View window
F4	Opens the Properties window
Ctrl + Shift + E	Opens the Resource view window
Ctrl + Alt + X	Opens the Toolbar window
Shift + Alt + Enter	Takes you to Full Screen View
Alt+F8	Opens Macro Explorer window
F2	Opens Object Browser window
Ctrl + Alt + T	Opens Document Outline window
Ctrl + Alt + K	Opens Task List window
Ctrl + Alt + A	Opens Command window
Ctrl + Alt + O	Opens Output window
Ctrl + Alt + Y	Opens Find Symbol Results window
Ctrl + Alt + F	Lists Items under the Favorites Menu in your Internet Explorer
Ctrl + Shift + B	Builds your project
F5	Runs your Application
Ctrl + F5	Runs your Application without Debugging
Ctrl + Alt + E	Opens the Exceptions Dialog Box

F8	Used while Debugging Applications
Shift + F8	Used While Debugging Applications
Ctrl + B	Inserts a New Breakpoint
Ctrl + Shift + F9	Clears All Breakpoints
Ctrl + Alt + P	Opens the Processes Dialog box
Ctrl + T	Opens Customize Toolbox window
Ctrl + Shift + P	Runs Temporary Macro
Ctrl + Shift + R	Records Temporary Macro
Alt + F11	Opens Macros IDE
Ctrl + F1	Opens Dynamic Help window
Ctrl +Alt + F1	Opens Help window sorted by Contents
Ctrl + Alt + F2	Opens Help window sorted by Index
Ctrl + Alt + F3	Opens Help Search window
Shift + Alt + F2	Opens Index Results window
Shift + Alt + F3	Opens Search Results window

Windows Forms

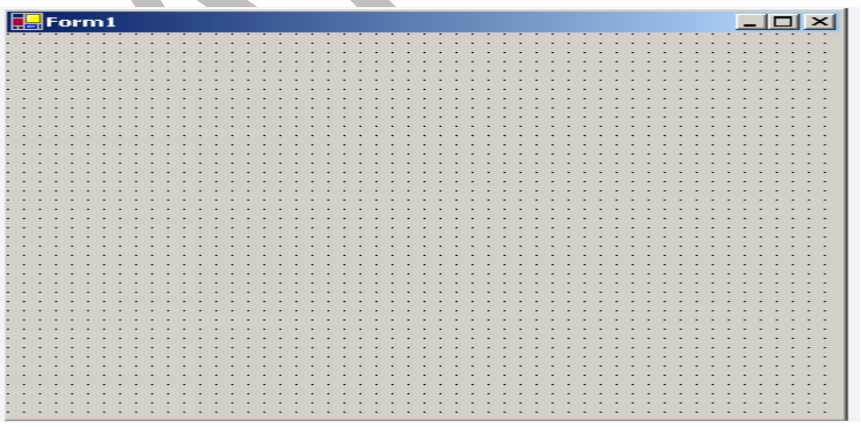
In Visual Basic its these Forms with which we work. They are the base on which we build, develop all our user interface and they come with a rich set of classes. Forms allow us to work visually with controls and other items from the toolbox. In VB .NET forms are based on the System.Windows.Forms namespace and the form class isSystem.Windows.Forms.Form. The form class is based on the Control class which allows it to share many properties and methods with other controls.

When we open a new project in Visual Basic the dialogue box that appears first is the one which looks like the image below. Since we are working with Windows Applications(Forms) you need to select WindowsApplication and click OK.



Once you click OK a new Form opens with the title, Form1, towards the top-left side of the form and maximize, minimize and close buttons towards the top right of the form. The whole form is surrounded with a border. The main area of the form in which we work is called the Client Area. It's in this client area we design the user interface leaving all the code to the code behind file. Forms also support events which let's the form know that something happened with the form, for example, when we double-click on the form, the Form load event occurs. VB .NET also supports forms to be inherited.

Image of a Windows Form.



Typically the Form looks like this in Code which is handled by the Framework.

```
Public Class Form1
Inherits System.Windows.Forms.Form
```

```
#Region " Windows Form Designer generated code "

Public Sub New()
MyBase.New()

'This call is required by the Windows Form Designer.
InitializeComponent()

'Add any initialization after the InitializeComponent() call

End Sub

'Form overrides dispose to clean up the component list.
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
If disposing Then
If Not (components Is Nothing) Then
components.Dispose()
End If
End If
MyBase.Dispose(disposing)
End Sub

'Required by the Windows Form Designer
Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Windows Form
Designer
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
<System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
'
'Form1
'
Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(496, 493)
Me.Name = "Form1"
Me.Text = "Form1"

End Sub

#End Region

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

End Sub
End Class
```

Windows Form Properties

Below are the properties of a Windows Form. Properties displayed below are categorized as seen in the properties window.

Appearance Properties	Description
BackColor	Gets/Sets the background color for the form
BackgroundImage	Get/Sets the background image in the form
Cursor	Gets/Sets the cursor to be displayed when the user moves the mouse over the form
Font	Gets/Sets the font for the form
ForeColor	Gets/Sets the foreground color of the form
FormBorderStyle	Gets/Sets the border style of the form
RightToLeft	Gets/Sets the value indicating if the alignment of the control's elements is reversed to support right-to-left fonts
Text	Gets/Sets the text associated with this form
Behavior Properties	**Description**
AllowDrop	Indicates if the form can accept data that the user drags and drops into it
ContextMenu	Gets/Sets the shortcut menu for the form
Enabled	Gets/Sets a value indicating if the form is enabled
ImeMode	Gets/Sets the state of an Input Method Editor
Data Properties	**Description**
DataBindings	Gets the data bindings for a control
Tag	Gets/Sets an object that contains data about a control
Design Properties	**Description**
Name	Gets/Sets name for the form
DrawGrid	Indicates whether or not to draw the positioning grid
GridSize	Determines the size of the positioning grid
Locked	Gets/Sets whether the form is locked
SnapToGrid	Indicates if the controls should snap to the positioning

	grid
Layout Properties	**Description**
AutoScale	Indicates if the form adjusts its size to fit the height of the font used on the form and scales its controls
AutoScroll	Indicates if the form implements autoscrolling
AutoScrollMargin	The margin around controls during auto scroll
AutoScrollMinSize	The minimum logical size for the auto scroll region
DockPadding	Determines the size of the border for docked controls
Location	Gets/Sets the co-ordinates of the upper-left corner of the form
MaximumSize	The maximum size the form can be resized to
MinimumSize	The minimum size the form can be resized to
Size	Gets/Sets size of the form in pixels
StartPosition	Gets/Sets the starting position of the form at run time
WindowState	Gets/Sets the form's window state
Misc Properties	**Description**
AcceptButton	Gets/Sets the button on the form that is pressed when the user uses the enter key
CancelButton	Indicates the button control that is pressed when the user presses the ESC key
KeyPreview	Determines whether keyboard controls on the form are registered with the form
Language	Indicates the current localizable language
Localizable	Determines if localizable code will be generated for this object
Window Style Properties	**Description**
ControlBox	Gets/Sets a value indicating if a control box is displayed
HelpButton	Determines whether a form has a help button on the caption bar
Icon	Gets/Sets the icon for the form
IsMdiContainer	Gets/Sets a value indicating if the form is a container for

	MDI child forms
MaximizeBox	Gets/Sets a value indicating if the maximize button is displayed in the caption bar of the form
Menu	Gets/Sets the MainMenu that is displayed in the form
MinimizeBox	Gets/Sets a value indicating if the minimize button is displayed in the caption bar of the form
Opacity	Determines how opaque or transparent the form is
ShowInTaskbar	Gets/Sets a value indicating if the form is displayed in the Windows taskbar
SizeGripStyle	Determines when the size grip will be displayed for the form
TopMost	Gets/Sets a value indicating if the form should be displayed as the topmost form of theapplication
TransparencyKey	A color which will appear transparent when painted on the form

Working with Forms

Well, let's now start working with Forms. When you open a new form you can have a look at the default properties of the form by selecting View->Properties Window or by pressing F4 on the keyboard. The properties window opens with default properties set to form by the software.

Briefly on Properties (Categorized):

Appearance

Appearance section of the properties window allow us to make changes to the appearance of the form. With the help of appearance properties we can have a background color, background image for the entire form, set the border style for the form from a predefined list, change the cursor, set the font for the text on the form and so on.

Behavior

Notable Behavior property is the enabled property which lets us enable/disable the form by setting the property to True/False.

Layout

Layout properties are all about the structure of the form. With these properties we can set the location of the form, maximum size of the form, minimum size of the form, exact size of the form with the size property when designing. Note the property start position, this property

lets you specify the location of the form where it should appear when you run the application which you can select from a predefined list.

Window Style

The notable property under this is the ControlBox property which by default it is set to True. Setting the property to False makes the minimize, maximize and cancel buttons on the top right side of the form invisible.

Form Event

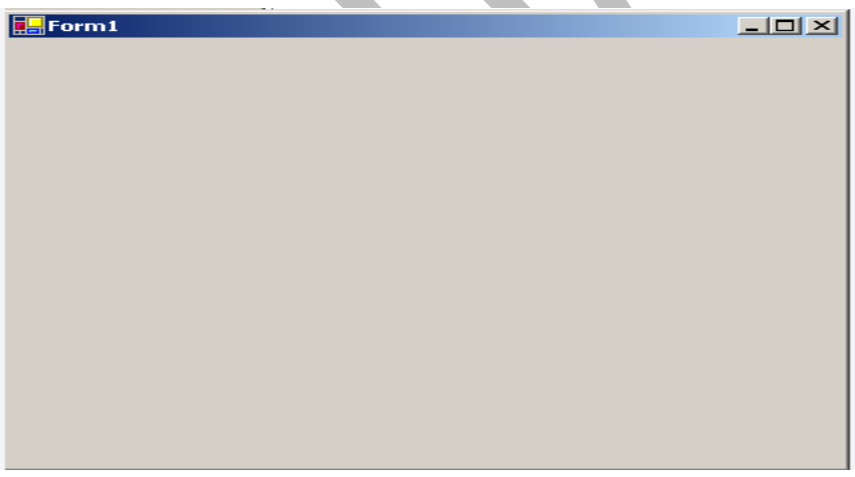
The default event of a form is the load event which looks like this in code:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs)_
Handles MyBase.Load

End Sub
```

You can write code in the load event of the form just like you write for all other controls.

An Example:

You can run the Form by selecting Debug->Start from the main menu or by pressing F5 on the keyboard. When you run a blank form with no controls on it, nothing is displayed. It looks like the image below.



Now, add a TextBox and a Button to the form from the toolbox. The toolbox can be selected from
View->ToolBox on the main menu or by holding Ctrl+Alt+X on the keyboard. Once adding the TextBox and Button is done, run the form. The output window displays a TextBox and a Button. When you click the Button nothing happens. We need to write an event for the Button stating something should happen when you click it. To do that get back to design view and double-click on the Button. Doing that opens an event handler for the Button where you specify what should happen when you click the button. That looks like this in code.

```
Public Class Form1
Inherits System.Windows.Forms.Form
#Region " Windows Form Designer generated code "
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs)_
Handles MyBase.Load
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click
End Sub
End Class
```

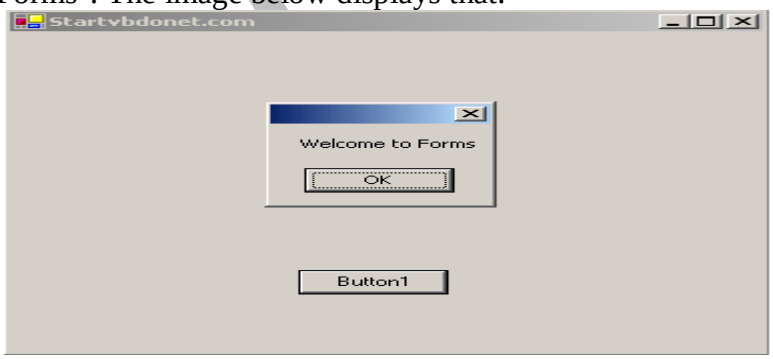
Place the code **TextBox1.Text="Welcome to Forms"** in the Click event of the Button and run the application. When you click the Button the output "Welcome to Forms" is displayed in the TextBox.

Alternatively, you can also use the MsgBox or MessageBox functions to display text when you click on the Button. To do that place a Button on the form and double-click on that to open it's event. Write this line of code, MsgBox("Welcome to Forms") or MessageBox.Show("Welcome to Forms").

It looks like this in code.

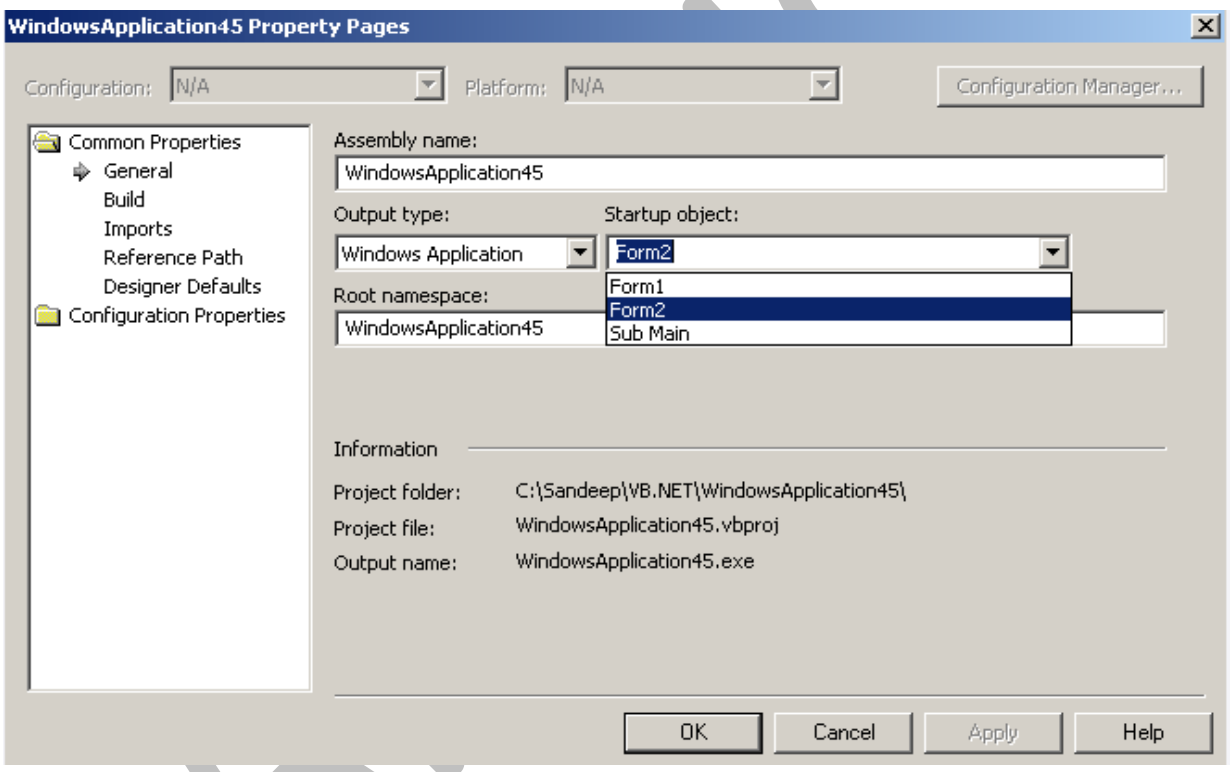
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click
MsgBox("Welcome to Forms")
End Sub

or

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click
MessageBox.Show("Welcome to Forms")
End Sub
```

When you run the form and click the Button, a small message box displays, "Welcome to Forms". The image below displays that.



Adding a New Form to the Project

You can add a new form to the project with which you are working. To do that, select File->Add New Item from the main menu which displays a window asking you what to add to the project. Select Windows Form from the window and click Open to add a new form to the project. Alternatively, you can add a new form with the Solution Explorer. To add a new form with the Solution Explorer, right-click on the project name in Solution Explorer and Select Add->Add Windows Form. Once you finish adding a new form, if you want the form which you added to be displayed when you run the application you need to make some changes. You need to set the new form as Startup Object. To do that, right-click on the project name in Solution Explorer window and select Properties which displays the Property Pages window. On this window click the drop-down box which is labeled as Startup Object. Doing that displays all the forms available in the project. It looks like the image below.



Select the form which you want to be displayed when you run the application and click Apply. Now, when you run the application, the form you assigned as Startup object will be displayed.

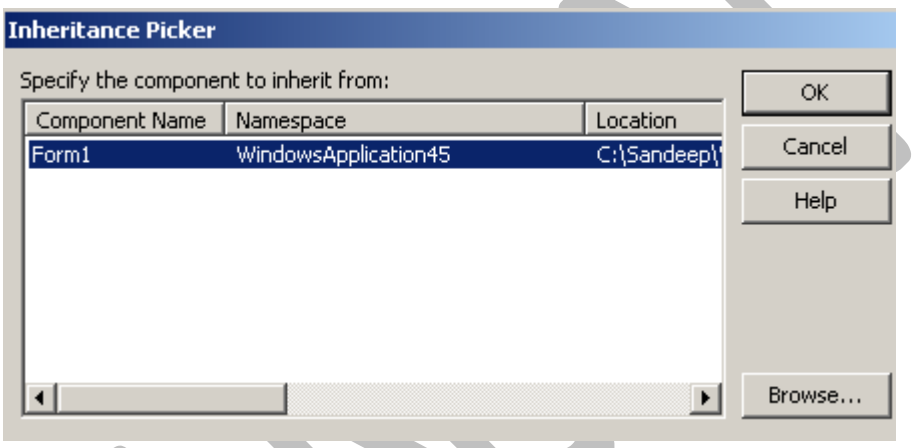
Working with Multiple Forms

Let's see how we can work with more than one form/open a new form. Say, for example, we want to open a new form (Form2) when a button in the current form (Form1) is clicked. To do that, open a new project by selecting File->New->Project->Visual Basic->WindowsApplication. This adds a form (Form1) to the application. Add a new form(Form2) by selecting File->Add New Item->Windows Form. Also, drag a button (Button1) onto Form1. We want to open Form2 when the button in Form1 is clicked. Unlike earlier versions of Visual Basic, VB .NET requires us to refer to Form2 in Form1 in order to open it i.e creating an object of Form2 in Form1. The code for that looks like this:

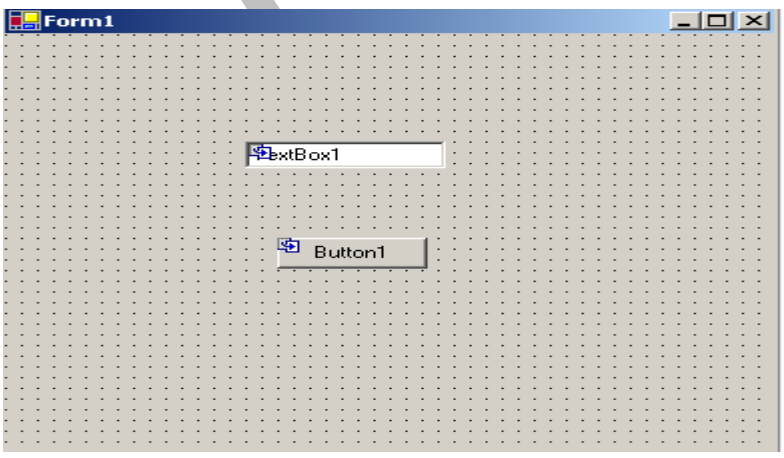
```
Public Class Form1 Inherits System.Windows.Forms.Form
Dim other As New Form2()
'Creating a reference to Form2
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
other.Show()
End Sub
```

Visual Inheritance with Forms

As we know, Inheritance allows us to derive one class from another. VB. NET allows us to inherit one form from another. Let's see how we can inherit a new form from an existing form. Say, we have a form named Form1 with some controls on it and we want to inherit a new form from that form. To derive a new form from the existing one select Project->Add Inherited Form from the main menu. Doing that opens the Add New Item window. Double-click on the Inherited Form Icon in the templates box to open the Inheritance Picker. Inheritance Picker window looks like the image below.



Select Form1 in the picker and click OK. Clicking OK adds a new form, Form2, which is derived from Form1. Everything on Form2 including the title is copied from Form1. The only difference between form1 and form2 is, the controls on Form2 come with a special icon at the upper left corner which indicates that the form is inherited and the controls are locked. The image below displays a Inherited Form.



Inheriting a Form from Other Project

You can also inherit a form from other project. To inherit a form from other project, navigate to the project containing the form you want using the browse button in the Inheritance Picker dialog, click the name of the DLL file containing the form and click Open. This returns to the inheritance Picker dialog box where the selected project is now listed. Choose the appropriate form and click OK. A new inherited form is added to your project.

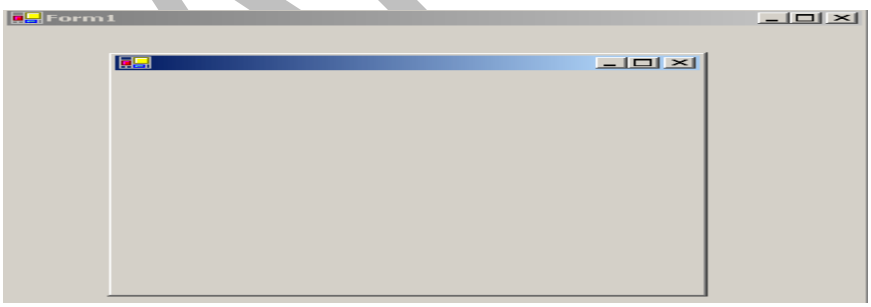
Owned Forms, **InputBox**

Owned Forms

Visual Basic also allows us to create owned forms. An owned form is tied to the form that owns it. If you minimize the owner form, the owned form will also be minimized, if you close the owner form, the owned form will be closed and so on. Owned Forms are added to a form with the AddOwnedForm method and removed with theRemoveOwnedForm method. The following code demonstrates the creation of owned forms. Drag a Button from the toolbox onto the form (form1). When you click thebutton, Form1 will make an object of the Form class into an owned form and displays it. The code for that looks like this:

```
Public Class Form1 Inherits System.Windows.Forms.Form
Dim OwnedForm1 As New Form()

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button2.Click
Me.AddOwnedForm(OwnedForm1)
'adding an owned form to the current form
OwnedForm1.show()
'displaying the owned form
End Sub
```

The image below displays output from above code.

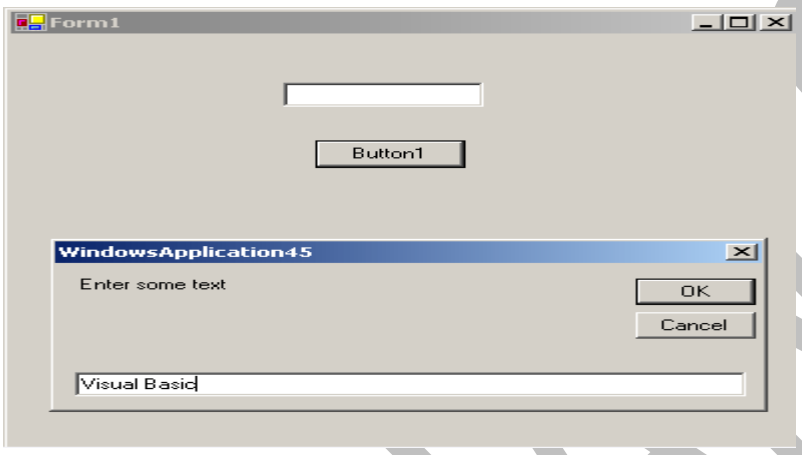


InputBox Function

InputBox function gets a string of text entered by the user. They are similar to the JavaScript prompt windows that ask us to enter some text and performs some action after the OK button is clicked. In Visual Basic Input boxes let you display a prompt, read the text entered by the user and returns a string result. The following code demonstrates InputBox function. Drag a Button and TextBox control from the toolbox onto the form. When you click the Button,

InputBox asks to enter some text and after you click OK it displays the text you entered in the TextBox. The image below displays output.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
Dim s As String = InputBox("Enter some text")
'storing the text entered in a string
TextBox1.Text = s
'displaying string in the textbox
End Sub
```



Anchoring and Docking Controls

Anchoring and Docking is used to make controls cover the whole client area of a form.

Anchoring

Anchoring is used to resize controls dynamically with the form. If you are designing a form that the user can resize at run time then the controls on your form should resize and reposition properly. The Anchor property defines an anchor position for the control. When a control is anchored to a form and the form is resized, the control maintainsthe distance between the control and the anchor positions.

Anchoring a control

To anchor a control, first select the control. In the Properties window select Anchor property. Selecting that displays an editor that shows a cross. To set an anchor, click the top, left, right, or bottom section of the cross. By default controls are anchored to the top and left side. To clear a side of the control that has been anchored click that side of the cross. When your run the form the control resizes to remain positioned at the same distance from the edge of the form. The distance from the anchored edge always remains the same as the distance defined when the control is positioned during design time. The image below demonstrates that.

Docking

When you dock a control, it adheres to the edges of its container (form). To dock a control, select the Dock property of that control from the properties window. Selecting the dock property opens up a small editor from which you can select to which side on the form should the control be docked.

Docking a control

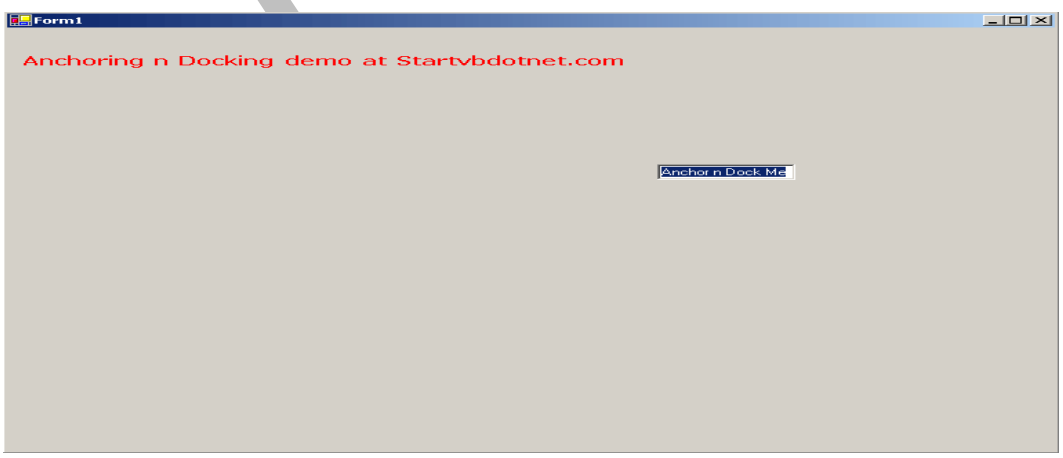
Select the control that you want to dock. In the Properties window select Dock property. An editor is displayed that shows a series of boxes representing the edges and thecenter of the form. Click the button that represents the edge of the form where you want to dock the control. To fill the contents of the control's form or container control, click the center box. The control is automatically resized to fit the boundaries of the docked edge.

Working with Example

The following three images will demonstrate Anchoring and Docking. The image below is a form with a TextBox on it.

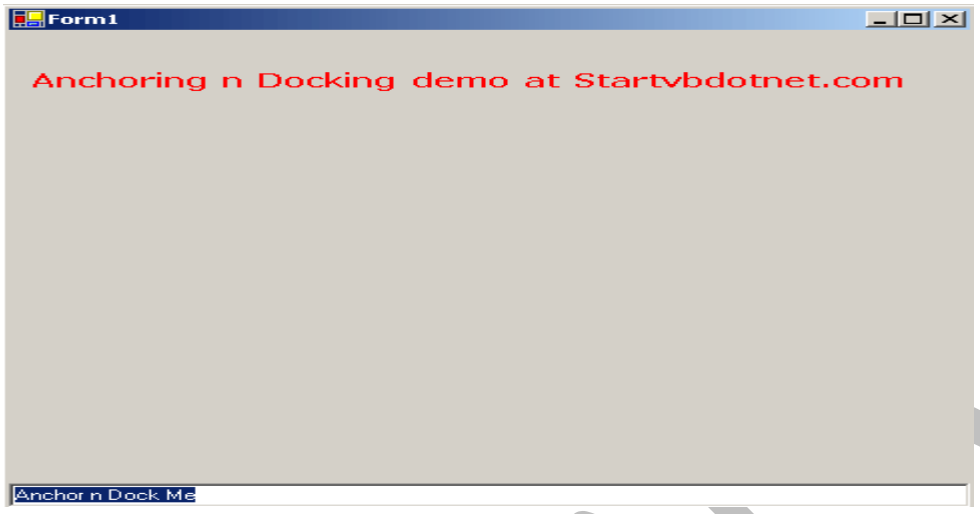
Anchoring

The image below displays the TextBox anchored to top-right side of the form. From the image you can view the TextBox maintaining the same distance from the top and right sides of the form when the form is increased in size.



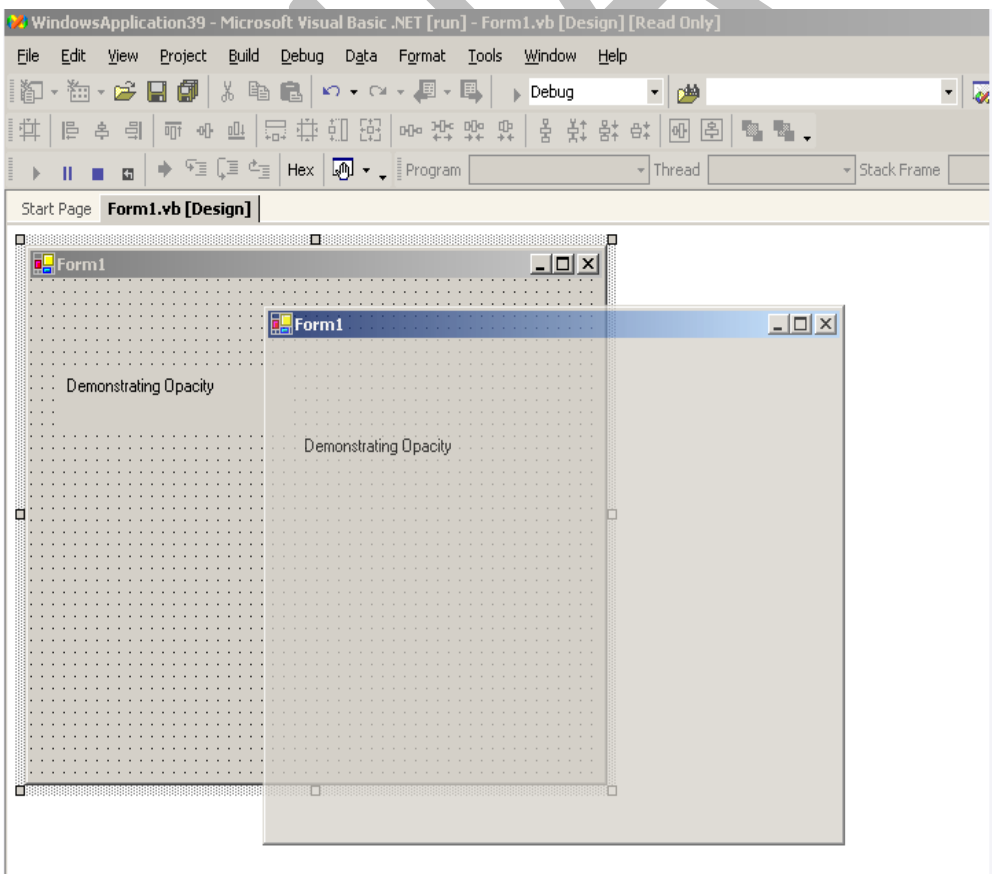
Docking

The image below displays the TextBox docked towards the bottom of the form.



Windows Forms Opacity

A beautiful feature of Windows Forms is Opacity. The Opacity property is used to make a form transparent. To set this property set the Opacity property to a value between 0 (complete transparency) and 100 (complete opacity) in the properties window. The Image below demonstrates Opacity property. The form is set to an opacity value of 75.



Transparent forms are only supported in Windows 2000 or later. Windows Forms will be completely opaque when run on older operating systems such as Windows 98/95 regardless of the value set for the Opacity property.

Handling Mouse Events in Forms

We can handle mouse events such as mouse pointer movements in Forms. The mouse events supported by VB .NET are as follows:

MouseDown: This event happens when the mouse pointer is over the form/control and is pressed
MouseEnter: This event happens when the mouse pointer enters the form/control
MouseUp: This event happens when the mouse pointer is over the form/control and the mouse button is released
MouseLeave: This event happens when the mouse pointer leaves the form/control
MouseMove: This event happens when the mouse pointer is moved over the form/control
MouseWheel: This event happens when the mouse wheel moves while the form/control has focus
MouseHover: This event happens when the mouse pointer hovers over the form/control

The properties of the MouseEventArgs objects that can be passed to the mouse event handler are as follows:

Button: Specifies that the mouse button was pressed
Clicks: Specifies number of times the mouse button is pressed and released
X: The X-coordinate of the mouse click
Y: The Y-coordinate of the mouse click
Delta: Specifies a count of the number of detents (rotation of mouse wheel) the mouse wheel has rotated

Working with an Example

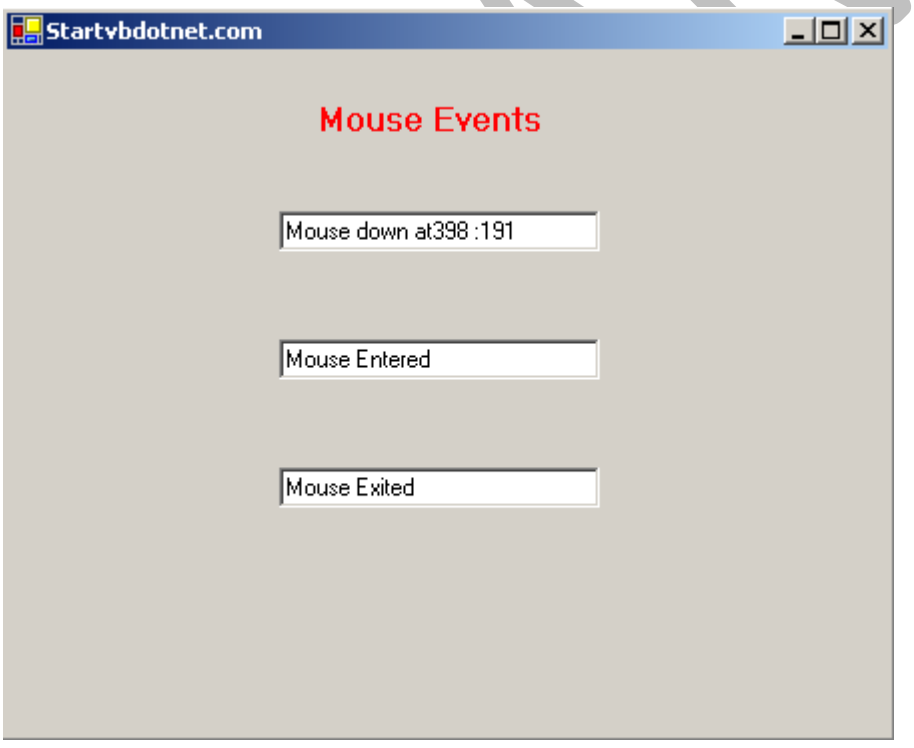
We will work with an example to check the mouse events in a form. We will be working with MouseDown, MouseEnter and MouseLeave events in this example. Drag three TextBoxes (TextBox1, TextBox2, TextBox3) onto the form. The idea here is to display the results of these events in the TextBoxes.

Code

```
Public Class Form1 Inherits System.Windows.Forms.Form
'Windows Form Designer Generated Code
Private Sub Form1_Mousedown(ByVal sender As System.Object, ByVal
e As _
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseDown
If e.Button = MouseButtons.Left Then
TextBox1.Text = "Mouse down at" + CStr(e.X) + " :" + CStr(e.Y)
'displaying the coordinates in TextBox1 when the mouse is pressed on the
form
End If
```

```
End Sub

Private Sub Form1_MouseEnter(ByVal sender As System.Object, ByVal
e_
As System.EventArgs) Handles MyBase.MouseEnter
TextBox2.Text = "Mouse Entered"
'displaying "mouse entered" in TextBox2 when the mouse pointer enters
the form
End Sub

Private Sub Form1_MouseLeave(ByVal sender As System.Object, ByVal
e_
As System.EventArgs) Handles MyBase.MouseLeave
TextBox3.Text = "Mouse Exited"
'displaying "mouse exited" in Textbox3 when the mouse pointer leaves the
form
End Sub

End Class
```

The image below displays the output.



Beep Function

The Beep Function in VB.NET can be used to make the computer emit a beep. To see how this works, drag a Button onto the form and place the following code in it's click event:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
```

```
As System.EventArgs) Handles Button1.Click
Beep()
End Sub
```

When you run the code and click the button your computer emits a beep.

MDI Applications

MDI (Multiple Document Interface) Application is an application in which we can view and work with several documents at once. Example of an MDI application is Microsoft Excel. Excel allows us to work with several documents at once. In contrast, SDI (Single Document Interface) applications are the applications which allows us to work with a single document at once. Example of a single document application is Microsoft Word in which only one document is visible at a time. Visual Basic .NET provides great support for creating and working with MDI applications. In general, MDI applications are mostly used by financial services organizations where the user needs to work withseveral documents at once.

Creating MDI Applications

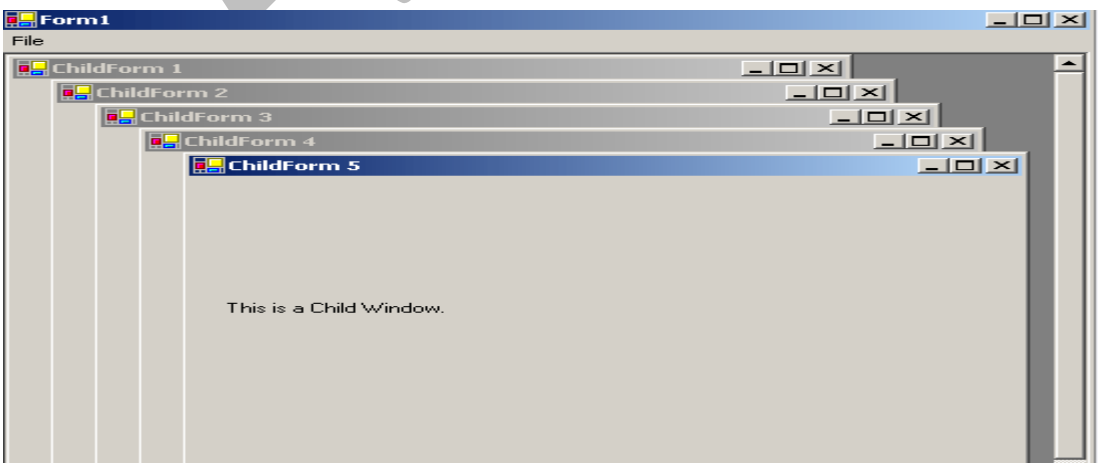
Let's create an MDI application. Open a new Windows Application in Visual Basic .NET. The application will open with a default form, Form1. Add another form, Form2 to thisapplication by right-clicking on the project name in Solution Explorer window and selecting Add->Add Windows Form. You can add some controls to Form2. For this applicationwe will make From1 as the MDI parent window and Form2 as MDI child window. MDI child forms are important for MDI Applications as users interact mostly through child forms. Select Form1 and in it's Properties Window under the Windows Style section, set the property IsMdiContainer to **True**. Setting it to true designates this form as an MDI container for the child windows. Once you set that property to true the form changes it's color. Now, from the toolbox drag a MainMenu component onto Form1. We will display child windows when a menu item is clicked. Name the top-level menu item to File with submenu items as New Child Window, Arrange Child Windows and Exit. The whole form should look like the image below.

With this application a new child window is displayed each time the New Child Window menu item is clicked, all child windows will be arranged when you click Arrange Child Windows menu item. To get the desired result, open the code designer window and paste the following code.

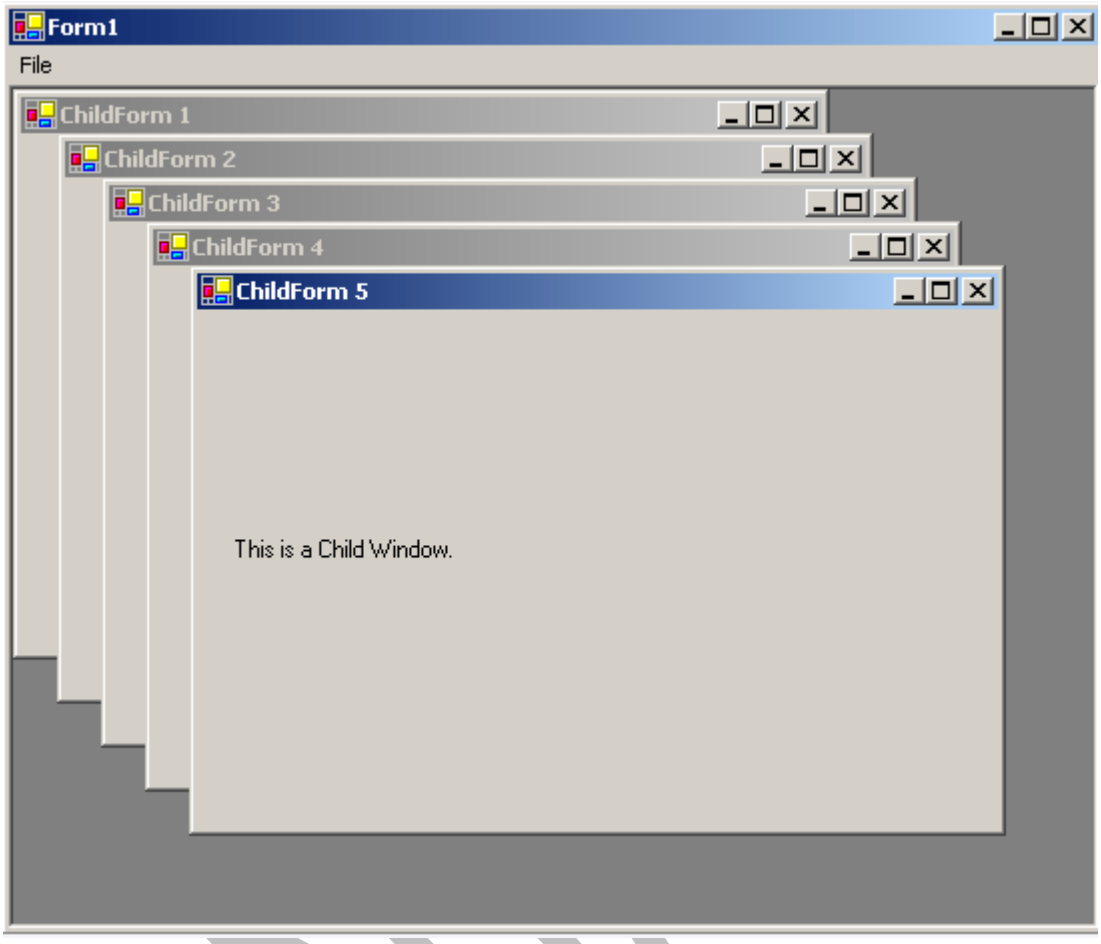
```
Public Class Form1 Inherits System.Windows.Forms.Form

Dim childForm As Integer = 0
Dim childForms(5) As Form2
'declaring an array to store child windows
'five child windows (Form2) will be displayed

#Region " Windows Form Designer generated code "
```

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles MyBase.Load
End Sub

Private Sub MenuItem2_Click(ByVal sender As System.Object,_
ByVal e As System.EventArgs) Handles MenuItem2.Click
childForm += 1
childForms(childForm) = New Form2()
childForms(childForm).Text = "ChildForm" & Str(childForm)
'setting title for child windows and incrementing the number with an array
childForms(childForm).MdiParent = Me
childForms(childForm).Show()
End Sub

Private Sub MenuItem3_Click(ByVal sender As System.Object,_
ByVal e As System.EventArgs) Handles MenuItem3.Click
Me.LayoutMdi(MdiLayout.Cascade)
'arranging child windows on the parent form with predefined LayoutMdi
method
'Different layouts available are, ArrangeIcons, Cascade, TileHorizontal,
TileVertical
End Sub

Private Sub MenuItem4_Click(ByVal sender As System.Object, ByVal_
e As System.EventArgs) Handles MenuItem4.Click
Me.Close()
'closing the application
End Sub

End Class
```

When you run the application and click "New Child Window" menu item, a new child window is displayed. Five child windows will be displayed as we declared an array of five in code. The image below displays the output.

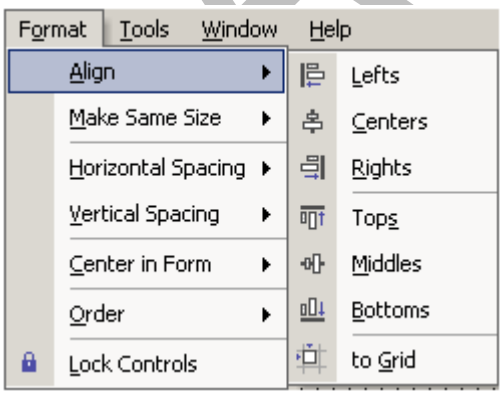


When you click on "Arrange Child Windows" menu item, all child windows are arranged. It looks like the image below.



Format Menu in VS .NET

The Format Menu available in Visual Studio .NET IDE allows us to align, layer and lock controls on a form. The Format menu provides many options for arranging controls.The Format Menu is shown in the image below.



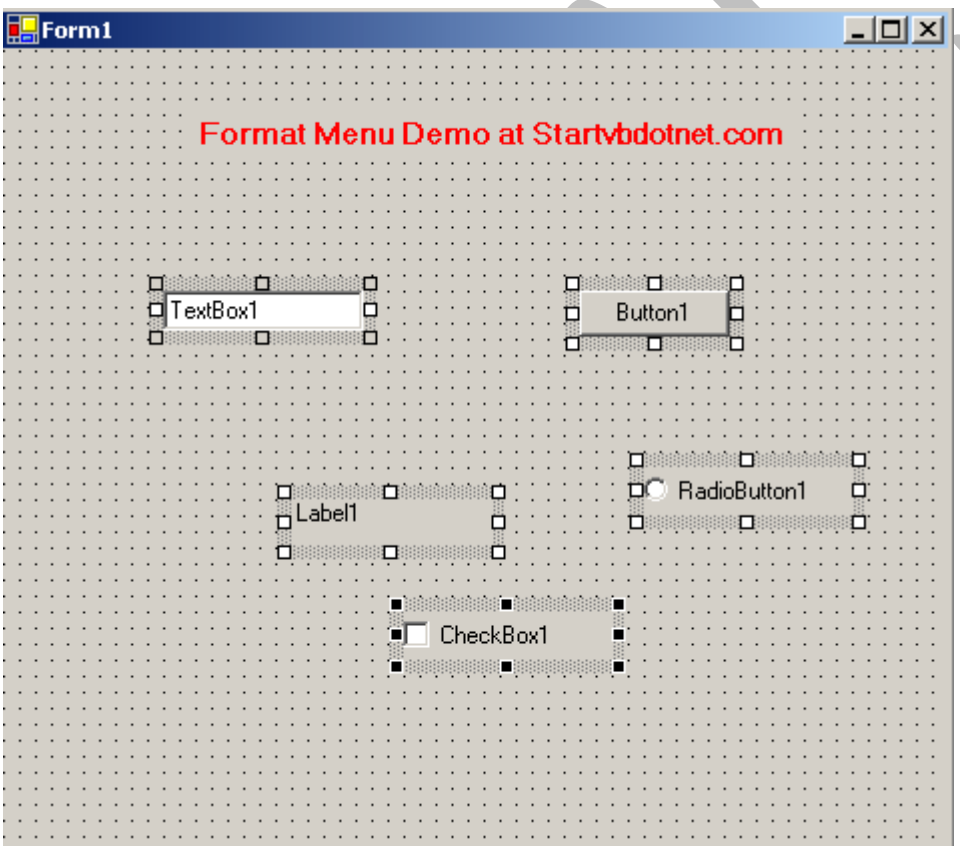
When using Format menu for arranging controls, we need to select the controls in such a way that the last control selected is the primary control to which other controls are aligned. The primary control has dark size handles and all other controls have light size handles.

The different options that are available under the Format menu are listed below.

Align: Aligns all controls with respect to the primary control
Make Same Size: Resizes multiple controls on a form
Horizontal Spacing: Increases horizontal spacing between controls
Vertical Spacing: Increases vertical spacing between controls
Center in Form: Centers the controls on form
Order: Layers controls on form
Lock Controls: Locks all controls on form

Aligning multiple controls on a Form

Let's work with an example. Open a new form and drag some controls onto it from the toolbox. Select the controls you want to align. The last control you select is the primary control to which all other controls align. On the Format menu point to align and then click any of the seven options available. The image below displays the controlsafter selection. From the image notice the CheckBox control with dark handles. That's the primary control.



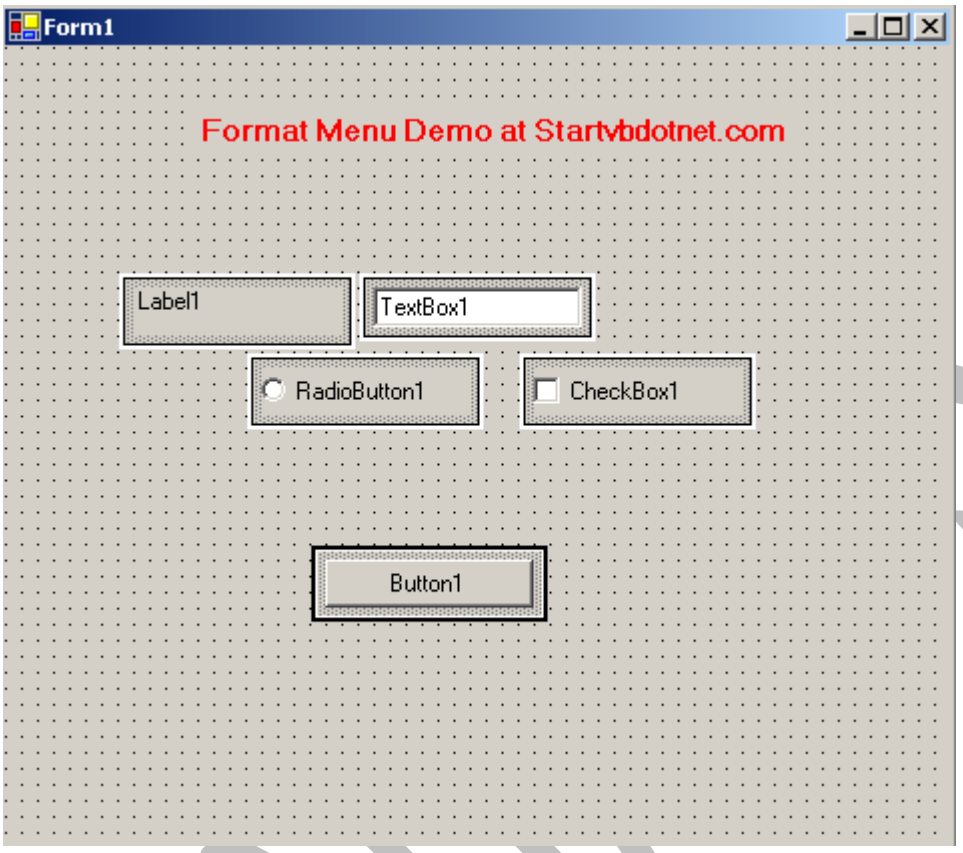
The image below displays controls that are aligned after selecting the option Lefts under Align from the Format menu. All the controls are left-aligned.

Make Same Size

The Make Same Size option under Format menu provides four options using which we can make all the controls same in size, width or height. The image below displayscontrols with the option width that makes all the controls same in width as the primary control.

Locking Controls

To lock all controls so that they cannot be moved accidentally, select Format menu and click lock controls. The image below displays that.



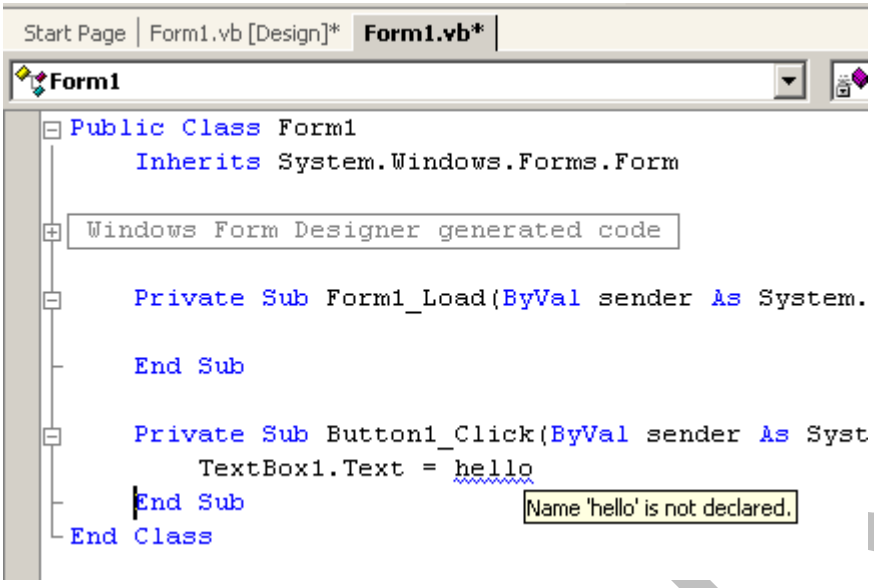
Debugging VB .NET Applications

In this section we will focus on some common issues while debugging our applications and we will take a close look at the Debug Menu Visual Studio .NET provides us. Errors in programming are inevitable. Everyone of us introduce errors while coding. Errors are normally referred to as bugs. The process of going through the code to identify the cause of errors and fixing them is called Debugging.

Types of Errors

Syntax Errors

The most common type of errors, syntax errors, occur when the compiler cannot compile the code. Syntax errors occur when keywords are typed incorrectly or an incorrectconstruct is parsed. The image below displays a syntax error in the code editor.

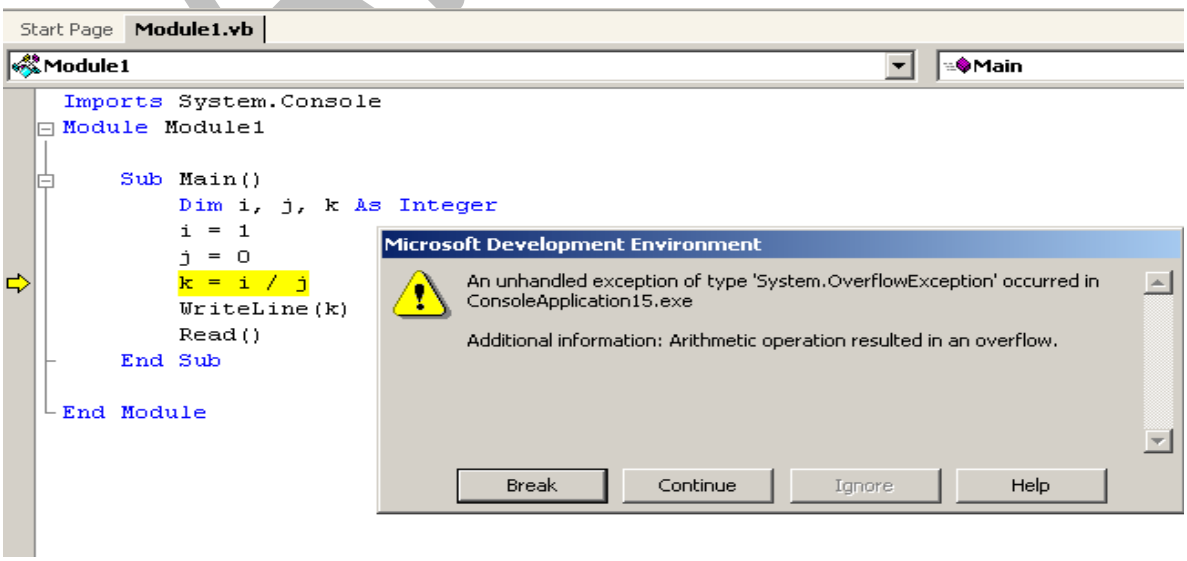


Fixing Syntax Errors

VS .NET allows us to easily identify syntax errors. When we build the project, syntax errors are automatically detected and underlined in code. Errors that are detected are also added to the Task List window. You can double-click any error in the Task List window and the cursor will immediately highlight that error in the code window. In most cases this is sufficient to correct the errors.

Run-Time Errors

Run-Time errors occur when the application attempts to perform a task that is not allowed. This includes tasks that are impossible to carry out, such as dividing by zero, etc. When a run-time error occurs, an exception describing the error is thrown. Exceptions are special classes that are used to communicate error states between different parts of the application. To handle run-time errors, we need to write code (normally, using Try...Catch...Finally) so that they don't halt execution of our application. The image below shows a run time error.



Logical Errors

Logical Errors occur when the application compiles and executes correctly, but does not output the desired results. These are the most difficult types of errors to trace because there might be no indications about the source of the error. The only way to detect and correct logical errors is by feeding test data into the application and analyzing the

Debugging VB .NET Applications

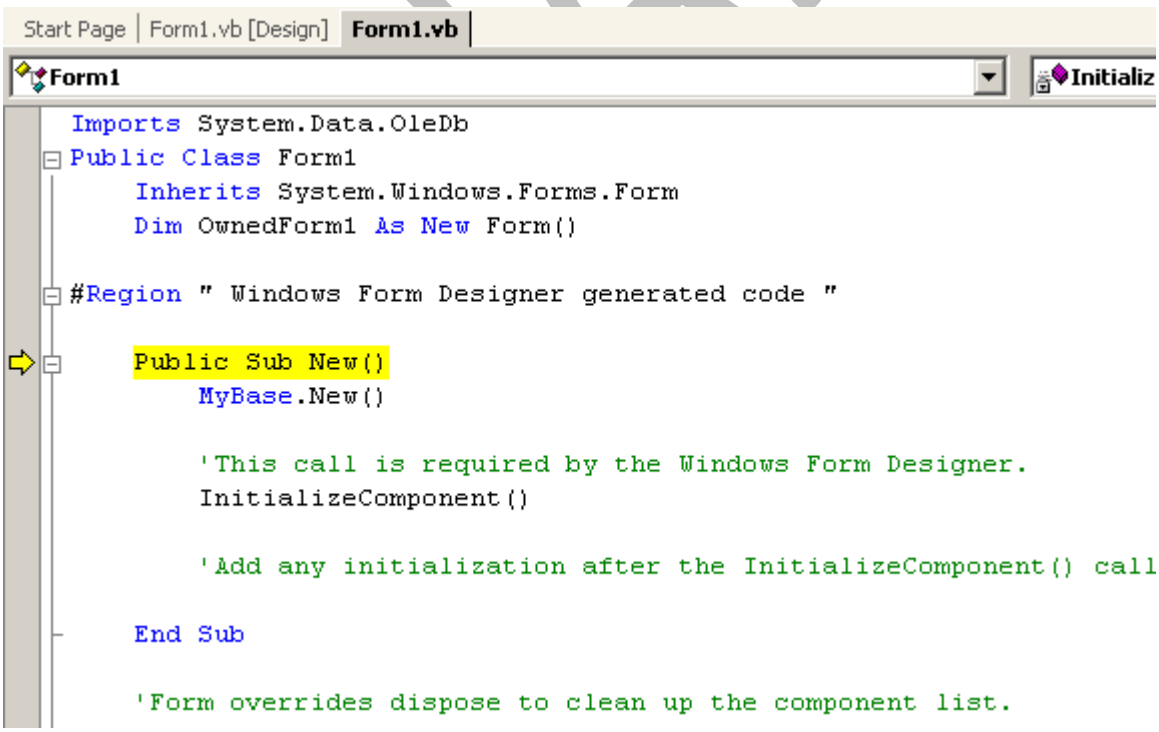
This section focuses on Breakpoints and the Debug Menu found in Visual Studio .NET, the items within it and their use.

Break Mode

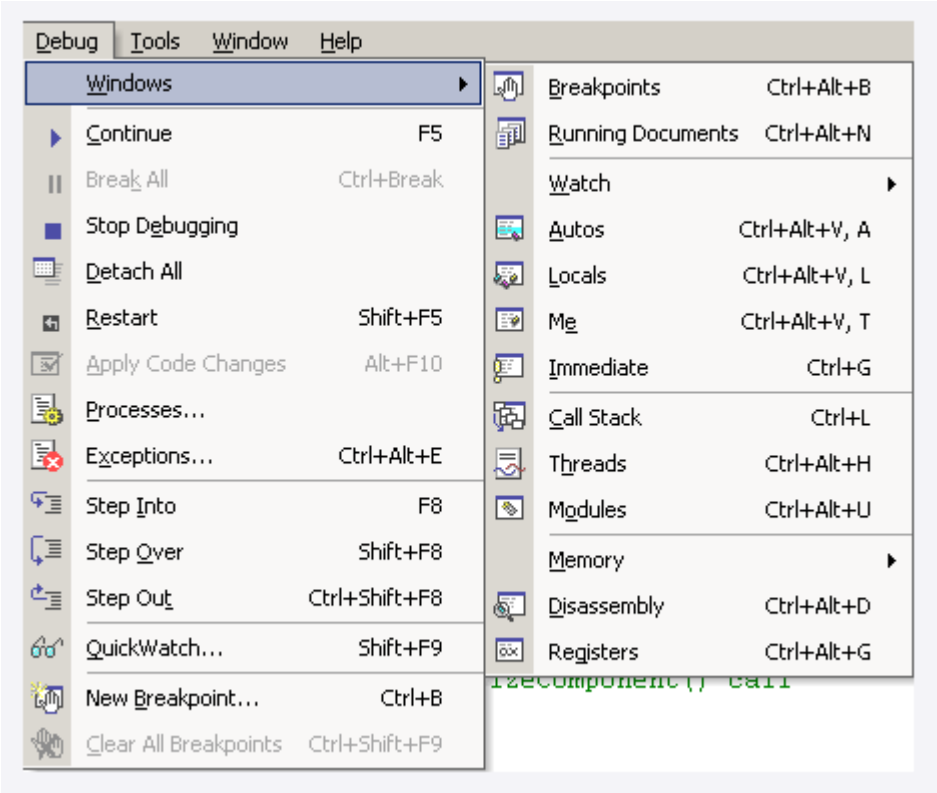
Break Mode allows us to halt code execution and execute code one line at a time. In break mode we can examine the values of application variables and properties. Visual Studio .NET enters break mode under any of the following circumstances:

- When we choose Step Into, Step Over or Step out from the Debug menu
- Execution reaches a line that contains an enabled breakpoint
- Execution reaches a Stop statement
- An unhandled exception is thrown

The image below shows the code designer window in Break Mode.



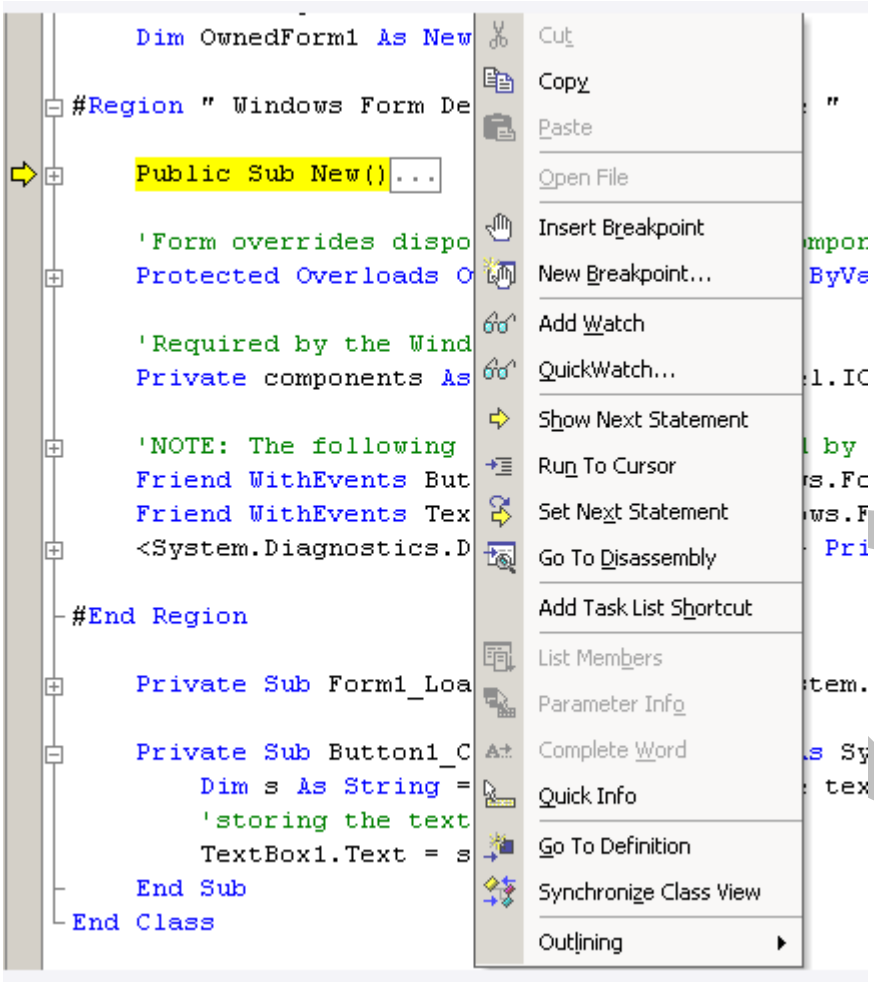
Once in Break mode we can use the debugging tools to examine our code. The image below shows the features on Debug menu.



Debug Menu Items from the above image summarized:

Windows: Opens a submenu that allows us to choose a debugging window to view.
Start/Continue: Runs the application in debug mode. In Break mode, this option continues program execution.
Break All: Causes program execution to halt and enter break mode at the current program line. You can choose continue to resume execution.
Stop Debugging: Stops the debugger and return to design mode in Visual Studio.
Detach All: Detaches the debugger from all processes it is debugging. This closes the debugger without halting program execution.
Restart: Terminates and restarts application execution.
Apply Code Changes: Works for C/C++ programming. Doesn't work for VB and C#.
Processes: Displays the processes window.
Exceptions: Displays the exception window.
Step Into: Runs the next executable line of code. If the next line calls a method, Step Into stops at the beginning of that method.
Step Over: Runs the next executable line of code. If the next line calls a method, Step Over executes that method and stops at the next line within the current method.
QuickWatch: Displays the QuickWatch window.
New Breakpoint: Displays the New Breakpoint Window.
Clear All Breakpoints: Removes all breakpoints from the application.
Disable All Breakpoints: Disables all breakpoints, but does not delete them.

There are some more debugging functions that can be accessible by right-clicking on an element in the code window and choosing a function from the pop-up menu. The image below displays those functions and summarizes them.



Insert Breakpoint: Inserts a breakpoint at the selected line.
New Breakpoint: Displays the new Breakpoint window. Identical to the Debug menu item of the same name.
Add Watch: Adds the selected expression to the watch window.
QuickWatch: Displays the QuickWatch window.
Show Next Statement: Highlights the next statement to be executed.
Run To Cursor: Runs program execution to the selected line.
Set Next Statement: Designates the selected line as the next line of code to be executed. The selected line should be in the current procedure.

Breakpoints

We can insert breakpoints at lines of code that will always cause the application to break in the debugger. Breakpoints are used to set lines of code that will halt code execution. There are four types of breakpoints as discussed below:

Function breakpoints: Causes the application to enter Break mode when the specified location in the function is reached.
File breakpoints: Causes the application to enter Break mode when the specified location in the file is reached.
Address breakpoints: Causes the application to enter Break mode when the specified

memory address is reached
Data breakpoints: Causes the application to enter Break mode when the value of a variable changes. Not available in VB and C#.

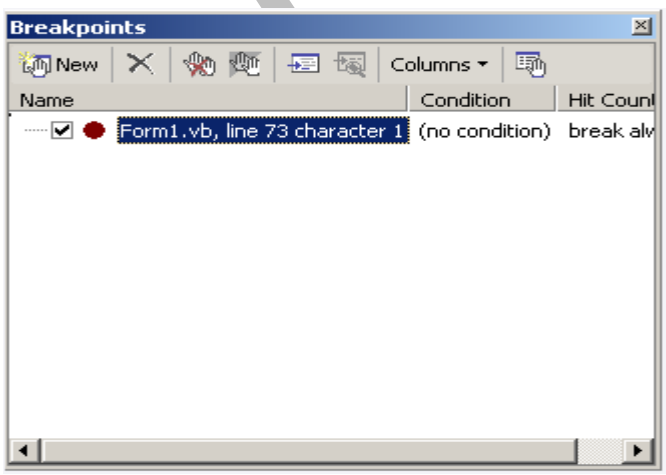
The image below displays the code designer when a breakpoint is inserted.

```
    'NOTE: The following procedure is required by the Wi
    Friend WithEvents Button1 As System.Windows.Forms.Bu
    Friend WithEvents TextBox1 As System.Windows.Forms.T
    <System.Diagnostics.DebuggerStepThrough()> Private S

#End Region

    Private Sub Form1_Load(ByVal sender As System.Object

    Private Sub Button1_Click_1(ByVal sender As System.C
        Dim s As String = InputBox("Enter some text")
        'storing the text entered in a string
        TextBox1.Text = s
    End Sub
End Class
```

When setting breakpoints with the New Breakpoint window we can attach conditions that determine whether or not execution halts when the breakpoint is reached. Pressing the Condition button displays the Breakpoint Condition window which allows us to designate an expression and causes the breakpoint to be active only if the breakpoint is true or if the expression changes.

Breakpoints Window

The Breakpoints Window lists all of the breakpoints in the project, like, where the breakpoint is located and any conditions attached to it. We can disable a breakpoint by clearing the check box next to it. The button in the Breakpoints window allows us to create a new breakpoint, delete a breakpoint or clear or disable all breakpoints. The Columns drop-down menu allows us to choose additional columns of information about the breakpoints. The image below displays a breakpoints window.



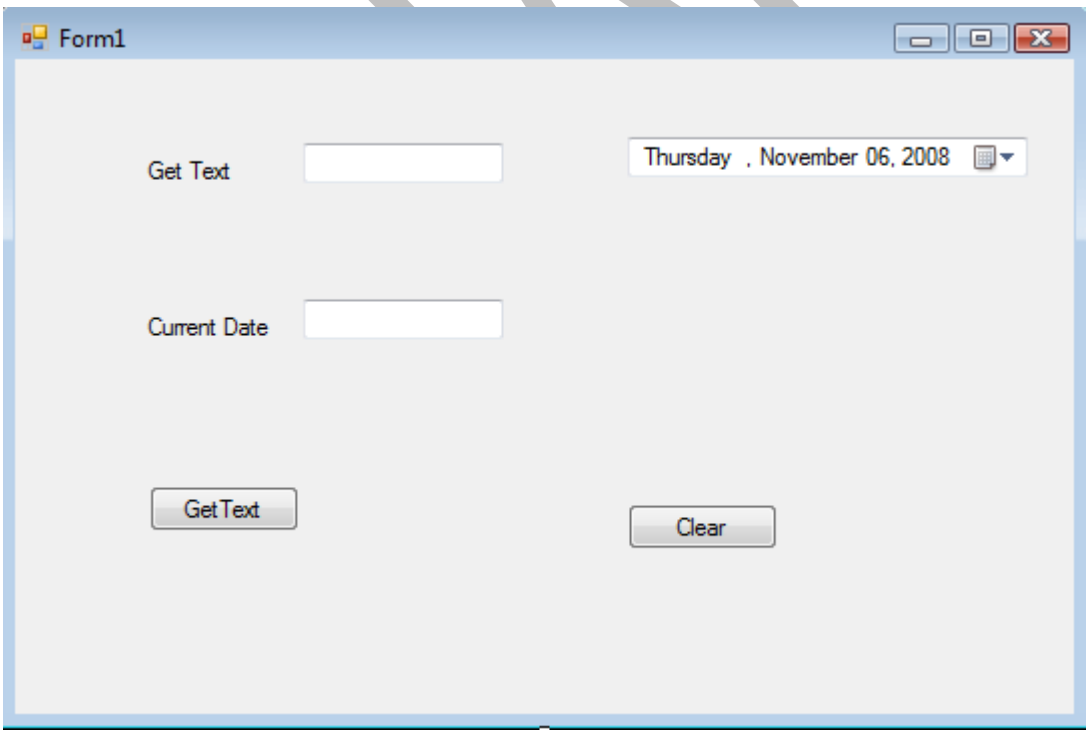
Removing and Disabling Breakpoints

We can remove a breakpoint by clicking on the circle like display towards the left side of the code window where the breakpoint is inserted. To disable a breakpoint, right-click the breakpoint in the code editor and select "Disable Breakpoint".

Sample Debugging Application

Whatever your project might be, debugging the application always helps you identify the cause of errors/bugs and helps you fix them. You might be working on a Strategic (new) project or Tactical (existing) project, debugging always comes handy. A simple scenario could be a project where the code was already written by other developers and you are finding it difficult to fix bugs. An example can be a huge windows application where on one of the forms you are clicking a button and this button might save a message along with the current system time. When you click this button you are able to save a message but recording a wrong time. In such a scenario you are suppossed to follow the debugging procedures and set breakpoints to find the code error. In this case you need to set a breakpoint at button_save event and press F11 to see the flow of the program and fix the bug. To understand debugging we will follow a simple example.

Here, we will work with a sample application to understand debgging. Open a new Windows Form and place 2 buttons, 2 labels, a datetimepicker control and 2 text boxes. The form in design mode should like like the image below.

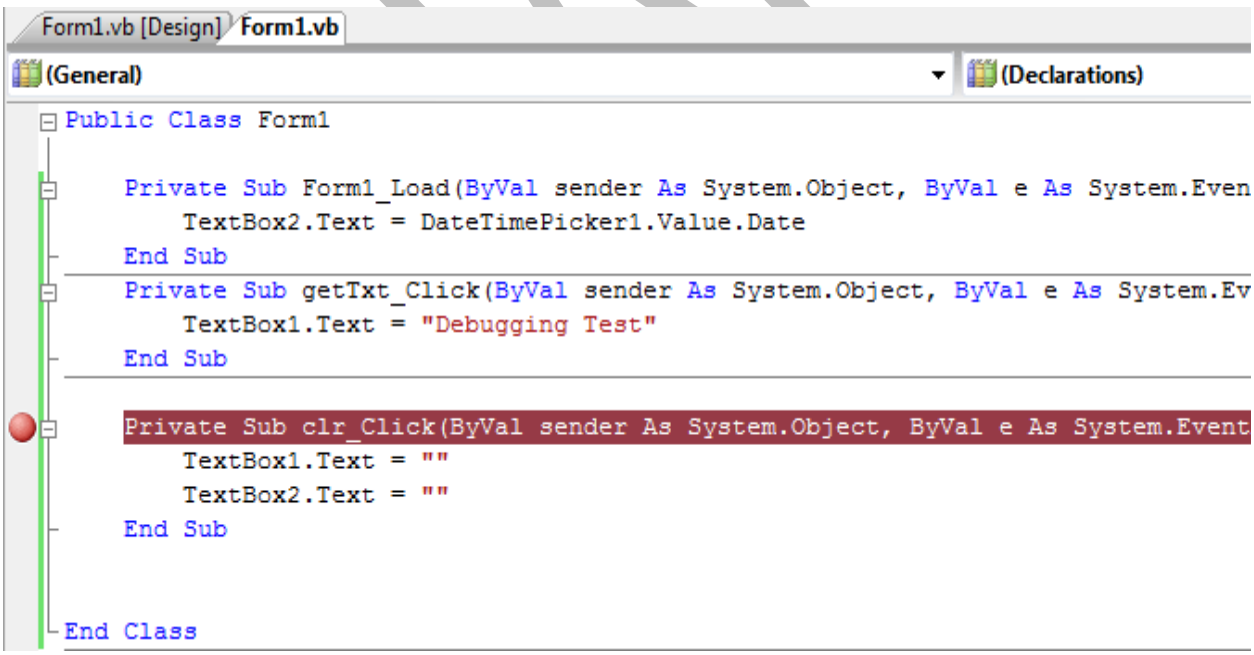


The codebehind for this form is as follows:

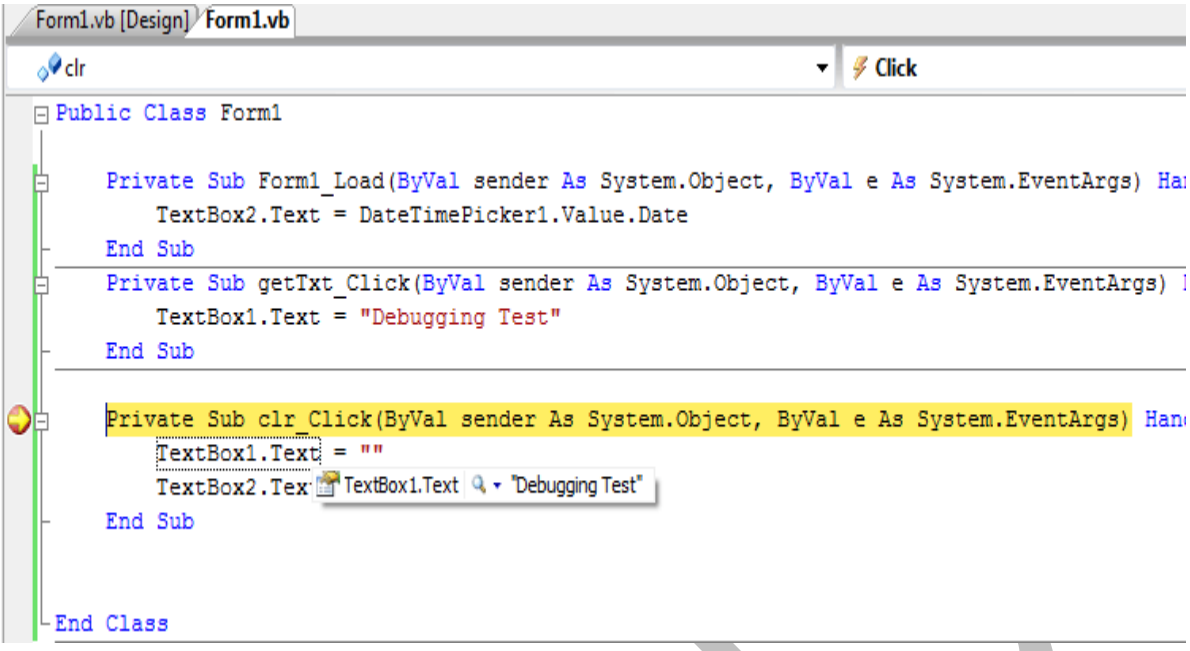
```
Public Class Form1

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
```

```
System.EventArgs)_
Handles MyBase.Load
TextBox2.Text = DateTimePicker1.Value.Date
End Sub

Private Sub getTxt_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)_
Handles getTxt.Click
TextBox1.Text = "Debugging Test"
End Sub

Private Sub clr_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)_
Handles clr.Click
TextBox1.Text = " "
TextBox2.Text = " "
End Sub

End Class
```

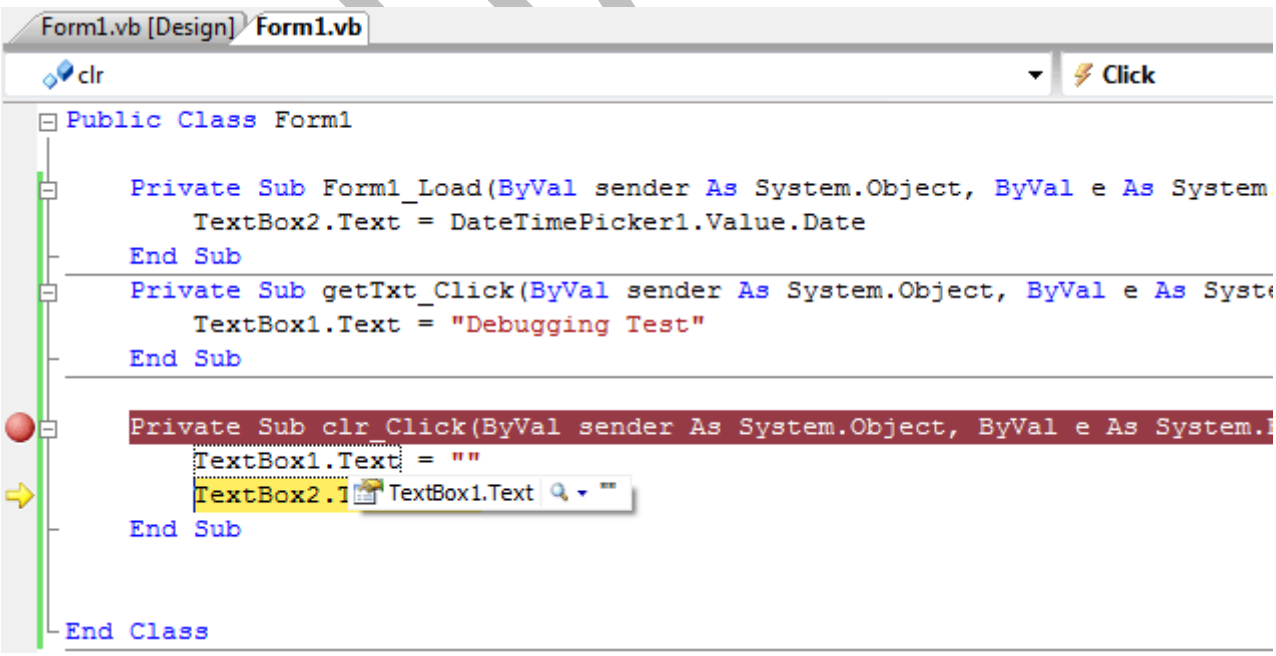
For the code listed above we will set a breakpoint at the clr_click event of the clear button in the code editor window. To set a breakpoint, point the mouse pointer towards the left of code and click it to see a red circle as shown in the image below.



Once you set the breakpoint, run the application by hitting F5. Current system date loads into textbox2. Click the get text button to fill textbox1 with text. Now, click the clear button to clear both the textboxes. Once you click the clear button, control shifts to the code editor, highlights the click event of the clear button with a yellow background and looks like the image below.



Now, when you place the mouse pointer on Textbox1.Text it shows the current value textbox1 is holding in it's text property which is "Debugging Text". Similarly, textbox2 shows the curent date when you place the mouse pointer on TextBox2.Text. Now, press the F11 (Step Into) key to see the control flow. The control shifts to TextBox1 and it still shows the text Debugging Test. Press F11 again and now the control shifts to TextBox2. Now, if you place the mouse pointer on TextBox1.Text it shows that the text property is set to null which means that the control has set the text property of TextBox1 to null. The image below shows that.



At this point, textbox2 still holds the current system date. Press F11 again and now the control shifts from textbox2 to End Sub. Now, if you place the mouse pointer on TextBox2.Text it shows it's text property as null. If you press F11 once again the control finishes End Sub and jumps back to the form which is running. What we saw now is the

passing of program flow when we set breakpoints and how the values change in the block while the control executes line by line.

With the above example we understood how debugging can help us find and fix errors. This was a small example but in real time applications it will be very helpful in finding and fixing some hard to find bugs. In a huge windows application there may be many forms and many of those forms might get inherited by other forms in a different solution or a different project in the same solution. Setting breakpoints at certain points on the form and running the application will help you to get to know how the program control flows, what method on one form is calling on another form and so on and helps you understand the logic n fix errors.

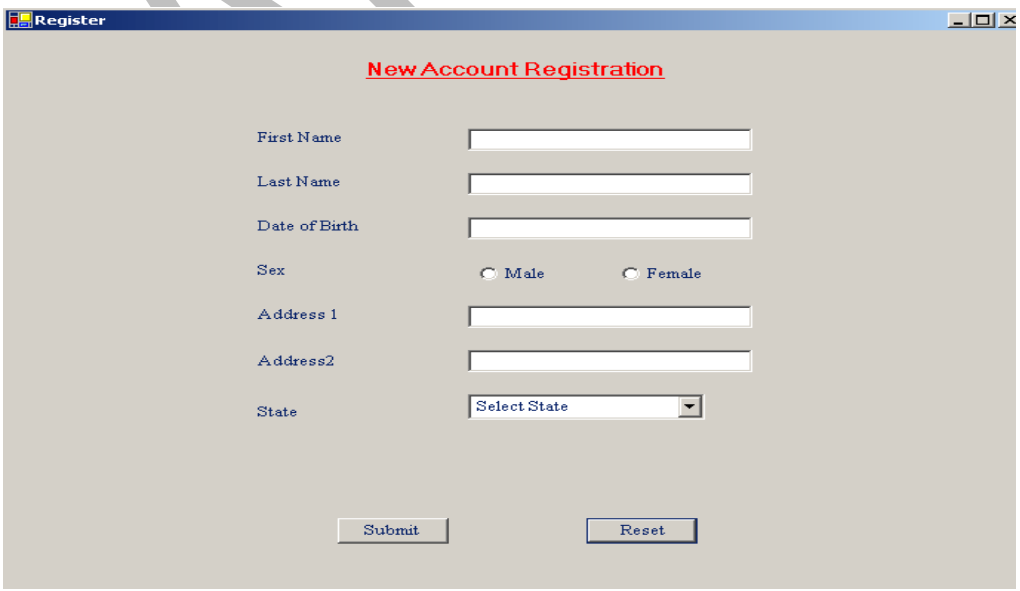
User Interface Design Principles

User Interface is a very important factor when we design our application (project). User interface provides a mechanism for end users to interact with the application. End users are called target audience. Designing a good user interface which is easy to use and understand is crucial for a successful application. If you already know the target audience for whom you are developing the application then designing becomes simple as you will already be familiar with their corporate colors and likes. Target audience, for example, can be the employee's of a firm for whom you will design the application. A well designed user interface makes it easy and simple for the target audience to understand and use. On the other hand, a poorly designed user interface will be hard to understand and use and can lead to distraction and frustration.

Good user interface is possible if it is designed keeping in mind the following four principles:

1) Simplicity
2) Positioning of Controls
3) Consistency
4) Aesthetics

The image below is an example of a good user interface design.



The above said four principles explained:

Simplicity

Simplicity is a key factor when designing a user interface. If a user interface looks crowded with controls then learning and using that application will be hard. Simplicity means, the user interface should allow the user to complete all the required tasks by the program quickly and easily. Also, program flow and execution should be kept on mind while designing. Try to avoid use of flashy and unnecessary images that distract the user. The simpler the user interface, the more friendly and easy it will be.

Positioning of Controls

Positioning of controls should reflect their importance. Say, for example, if you are designing an application that has a data-entry form with textboxes, buttons, radio buttons, etc. The controls should be positioned in such a way that they are easy to located and matches the program flow. Like, a submit button should be placed at the bottom of the form so that when the user enters all the data he can click it straight away. The image above is a perfect example of positioning of controls.

Consistency

The user interface should have a consistent look through out the application. The key to consistency lies during the design process. Before developing an application, we need to plan and decide a consistent visual scheme for the application that will be followed throughout. Using of particular fonts for special purposes, using of colors for headings, use of images, etc are all part of consistency.

Aesthetics

An application should project an inviting and pleasant user interface. The following should be considered for that.

Color

Use of color is one way to make the user interface attractive to the user. The color which you select for text and back-ground should be appealing. Care should be taken to avoid gaudy colors that are disturbing to the eye, for example, black text on a red back-ground.

Fonts

The fonts which you use for text should also be selected with care. Simple, easy-to-read fonts like Verdana, Times New Roman should be used. Try to avoid bold, strikeout text in most parts of the application. Use of bold, italics and other formatting should be limited to important text or headings.

Images

Images add visual interest to the application. Simple, plain images should be used wherever appropriate. Avoid using flashing images and images that are not necessarybut are used only for show off.

Shapes and Transparency

.NET Framework provides tools that allow us to create forms and controls with different levels of opacity. Apart from using traditional shapes like rectangles, etc, these tools also allow us to draw our own shapes which can provide some very powerful visual effects. User drawn shapes should be used only if the application requires it and care should be taken that the shapes which are drawn do not disturb the eye.

The image below is an example of a poorly designed user interface.

ADO .NET

Most applications need data access at one point of time making it a crucial component when working with applications. Data access is making the application interact with a database, where all the data is stored. Different applications have different requirements for database access. VB .NET uses ADO .NET (Active X Data Object) as it's data access and manipulation protocol which also enables us to work with data on the Internet. Let's take a look why ADO .NET came into picture replacing ADO.

Evolution of ADO.NET

The first data access model, DAO (data access model) was created for local databases with the built-in Jet engine which had performance and functionality issues. Next cameRDO (Remote Data Object) and ADO (Active Data Object) which were designed for Client Server architectures but, soon ADO took over RDO. ADO was a good architecture but as the language changes so is the technology. With ADO, all the data is contained in a recordset object which had problems when implemented on the network and penetrating firewalls. ADO was a connected data access, which means that when a connection to the database is established the connection remains open until the application is closed. Leaving the connection open for the lifetime of the application raises concerns about database security and network traffic. Also, as databases are becoming increasingly important and as they are serving more people, a connected data access model makes us think about its productivity. For example, an application with connected data access may do well when connected to two clients, the same may do poorly when connected to 10 and might be unusable when connected to 100 or more. Also, open database connections use system resources to a maximum extent making the system performance less effective.

Why ADO.NET?

To cope up with some of the problems mentioned above, ADO .NET came into existence. ADO .NET addresses the above mentioned problems by maintaining a disconnecteddatabase access model which means, when an application interacts with the database, the connection is opened to serve the request of the application and is closed as soon as the request is completed. Likewise, if a database is Updated, the connection is opened long enough to complete the Update operation and is closed. By keeping connections open for only a minimum period of time, ADO .NET conserves system resources and provides maximum security for databases and also has less impact on system performance. Also, ADO .NET when interacting with the database uses XML and converts all the data into XML format for database related operations making them more efficient.

The ADO.NET Data Architecture

Data Access in ADO.NET relies on two components: DataSet and Data Provider.

DataSet

The dataset is a disconnected, in-memory representation of data. It can be considered as a local copy of the relevant portions of the database. The DataSet is persisted in memory and the data in it can be manipulated and updated independent of the database. When the use of

this DataSet is finished, changes can be made back to the central database for updating. The data in DataSet can be loaded from any valid data source like Microsoft SQL server database, an Oracle database or from a Microsoft Access database.

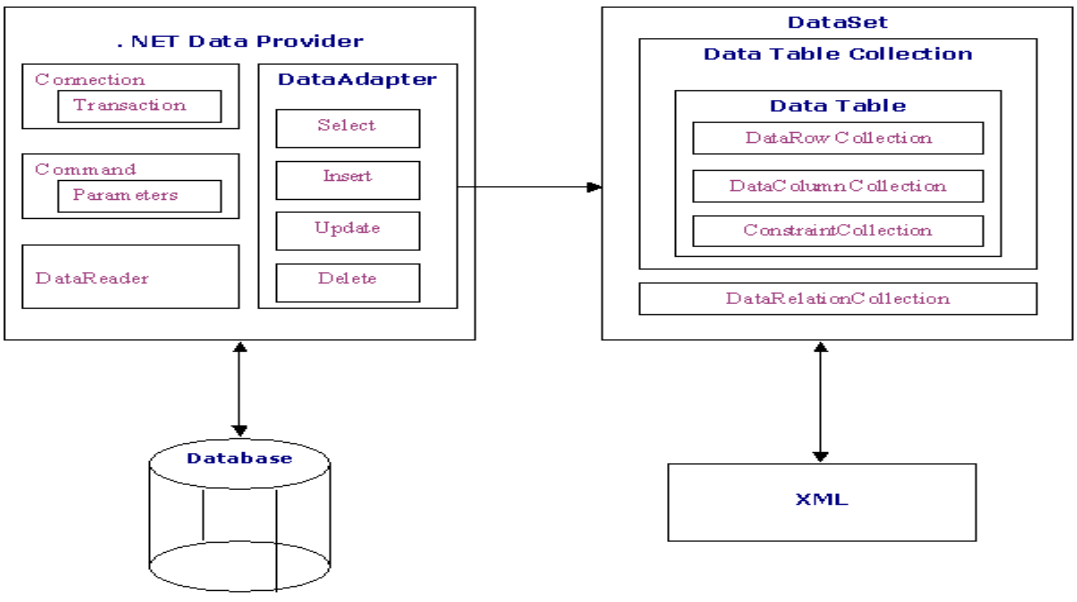
Data Provider

The Data Provider is responsible for providing and maintaining the connection to the database. A DataProvider is a set of related components that work together to provide data in an efficient and performance driven manner. The .NET Framework currently comes with two DataProviders: the SQL Data Provider which is designed only to work with Microsoft's SQL Server 7.0 or later and the OleDb DataProvider which allows us to connect to other types of databases like Access and Oracle. Each DataProvider consists of the following component classes:

The Connection object which provides a connection to the database
The Command object which is used to execute a command
The DataReader object which provides a forward-only, read only, connected recordset
The DataAdapter object which populates a disconnected DataSet with data and performs update

Data access with ADO.NET can be summarized as follows:

A connection object establishes the connection for the application with the database. The command object provides direct execution of the command to the database. If the command returns more than a single value, the command object returns a DataReader to provide the data. Alternatively, the DataAdapter can be used to fill the Dataset object. The database can be updated using the command object or the DataAdapter.



ADO .NET Data Architecture

Component classes that make up the Data Providers

The Connection Object

The Connection object creates the connection to the database. Microsoft Visual Studio .NET provides two types of Connection classes: the SqlConnection object, which is designed specifically to connect to Microsoft SQL Server 7.0 or later, and the OleDbConnection object, which can provide connections to a wide range of database types like Microsoft Access and Oracle. The Connection object contains all of the information required to open a connection to the database.

The Command Object

The Command object is represented by two corresponding classes: SqlCommand and OleDbCommand. Command objects are used to execute commands to a database across a data connection. The Command objects can be used to execute stored procedures on the database, SQL commands, or return complete tables directly. Command objects provide three methods that are used to execute commands on the database:

ExecuteNonQuery: Executes commands that have no return values such as INSERT, UPDATE or DELETE
ExecuteScalar: Returns a single value from a database query
ExecuteReader: Returns a result set by way of a DataReader object

The DataReader Object

The DataReader object provides a forward-only, read-only, connected stream recordset from a database. Unlike other components of the Data Provider, DataReader objects cannot be directly instantiated. Rather, the DataReader is returned as the result of the Command object's ExecuteReader method. The SqlCommand.ExecuteReader method returns a SqlDataReader object, and the OleDbCommand.ExecuteReader method returns an OleDbDataReader object. The DataReader can provide rows of data directly to application logic when you do not need to keep the data cached in memory. Because only one row is in memory at a time, the DataReader provides the lowest overhead in terms of system performance but requires the exclusive use of an open Connection object for the lifetime of the DataReader.

The DataAdapter Object

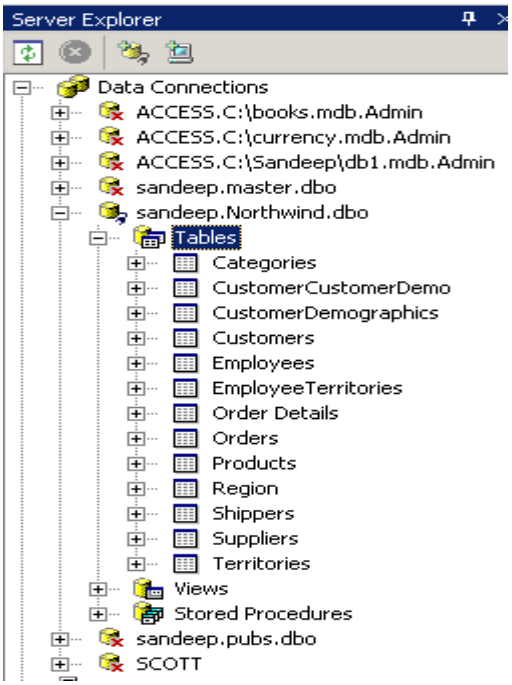
The DataAdapter is the class at the core of ADO .NET's disconnected data access. It is essentially the middleman facilitating all communication between the database and a DataSet. The DataAdapter is used either to fill a DataTable or DataSet with data from the database with it's Fill method. After the memory-resident data has been manipulated, the DataAdapter can commit the changes to the database by calling the Update method. The DataAdapter provides four properties that represent database commands:

SelectCommand
InsertCommand
DeleteCommand
UpdateCommand

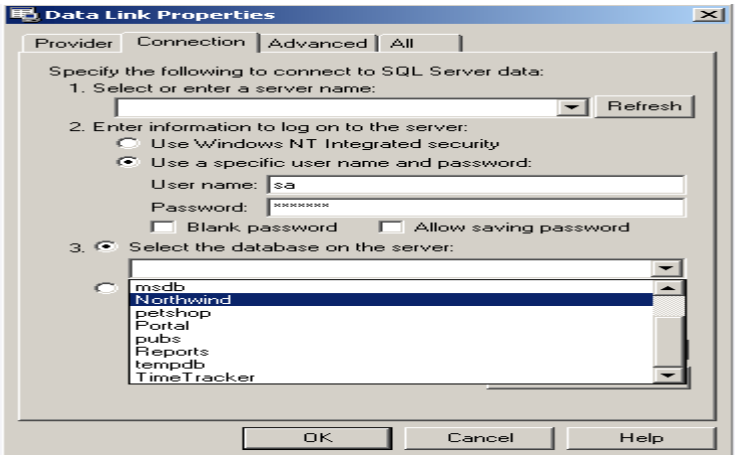
When the Update method is called, changes in the DataSet are copied back to the database and the appropriate InsertCommand, DeleteCommand, or UpdateCommand is executed.

Data Access with Server Explorer

Visual Basic allows us to work with databases in two ways, visually and code. In Visual Basic, Server Explorer allows us to work with connections across different data sourcesvisually. Lets see how we can do that with Server Explorer. Server Explorer can be viewed by selecting View->Server Explorer from the main menu or by pressing Ctrl+Alt+S on the keyboard. The window that is displayed is the Server Explorer which lets us create and examine data connections. The Image below displays the Server Explorer.

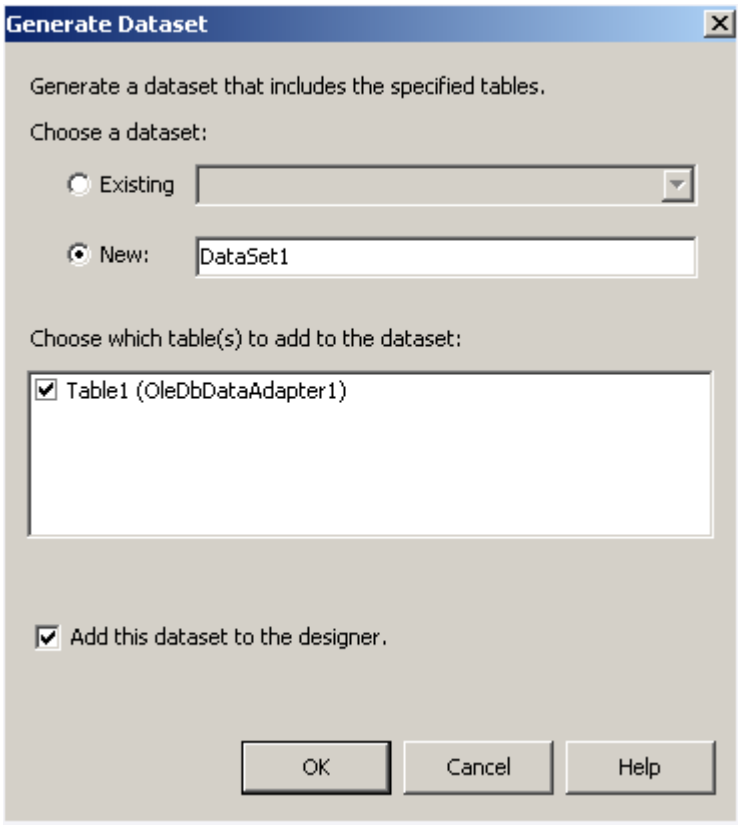


Lets start working with the Server Explorer. We will work with SQL Server, the default provider for .NET. We'll be displaying data from Customers table in sample Northwind database in SQL Server. First, we need to establish a connection to this database. To do that, right-click on the Data Connections icon in Server Explorer and select Add Connection item. Doing that opens the Data Link Properties dialog which allows you to enter the name of the server you want to work along with login name and password. The Data Link properties window can be viewed in the Image below.



Since we are working with a database already on the server, select the option "select the database on the server". Selecting that lists the available databases on the server, select Northwind database from the list. Once you finish selecting the database, click on the Test Connection tab to test the connection. If the connection is successful, the message "Test Connection Succeeded" is displayed. When connection to the database is set, click OK and close the Data Link Properties. Closing the data link properties adds a new Northwind database connection to the Server Explorer and this connection which we created just now is part of the whole Visual Basic environment which can be accessed even when working with other applications. When you expand the connection node ("+" sign), it displays the Tables, Views and Stored Procedures in that Northwind sample database. Expanding the Tables node will display all the tables available in the database. In this example we will work with Customers table to display its data.

Now drag Customers table onto the form from the Server Explorer. Doing that creates SQLConnection1 and SQLDataAdapter1 objects which are the data connection and data adapter objects used to work with data. They are displayed on the component tray. Now we need to generate the dataset that holds data from the data adapter. To do that select Data->Generate DataSet from the main menu or right-click SQLDataAdapter1 object and select generate DataSet menu. Doing that displays the generate Dataset dialogbox like the image below.



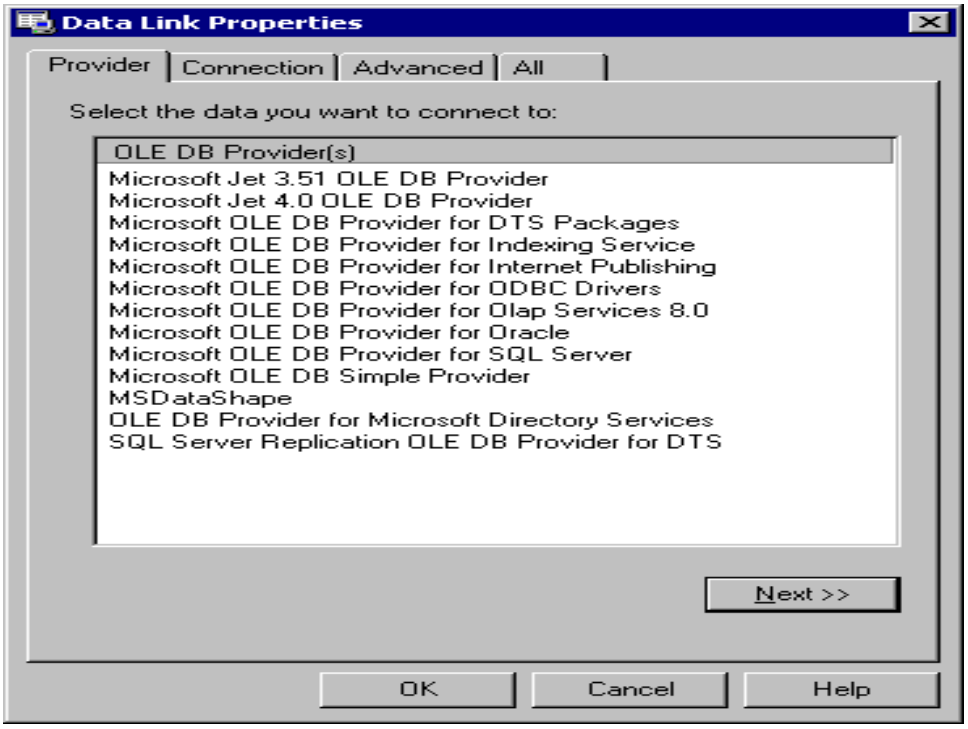
Once the dialogbox is displayed, select the radio button with New option to create a new dataset. Make sure Customers table is checked and click OK. Clicking OK adds a dataset, DataSet11 to the component tray and that's the dataset with which we will work. Now, drag a DataGrid from toolbox. We will display Customers table in this data grid. Set the data grid's DataSource property to DataSet11 and it's DataMember property to Customers. Next, we need to fill the dataset with data from the data adapter. The following code does that:

```
Private Sub Form1_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs)_
Handles MyBase.Load
DataSet11.Clear()
SqlDataAdapter1.Fill(DataSet11)
'filling the dataset with the dataadapter's fill method
End Sub
```

Once the application is executed, Customers table is displayed in the data grid. That's one of the simplest ways of displaying data using the Server Explorer window.

Microsoft Access and Oracle Database

The process is same when working with Oracle or MS Access but with some minor changes. When working with Oracle you need to select Microsoft OLE DB Provider for Oracle from the Provider tab in the DataLink dialog. You need to enter the appropriate Username and password. The Data Link Properties window can be viewed in the Imagebelow.



When working with MS Access you need to select Microsoft Jet 4.0 OLE DB provider from the Provider tab in DataLink properties.

Using DataReaders, SQL Server

In this section we will work with databases in code. We will work with ADO .NET objects in code to create connections and read data using the data reader. We will see how toconnect using our own connection objects, how to use the command object and so on. The namespace that needs to be imported when working with SQL Connections isSystem.Data.SqlClient. This section works with common database operations like insert, select, update and delete commands.

Working with SQL Server

When working with SQL Server the classes with which we work are described below.

The SqlConnection Class
The SqlConnection class represents a connection to SQL Server data source. We use OleDB connection object when working with databases other than SQL Server. Performance is the major difference when working with SqlConnections and OleDbConnections. Sql connections are said to be 70% faster than OleDb connections.

The SqlCommand Class
The SqlCommand class represents a SQL statement or stored procedure for use in a database with SQL Server.

The SqlDataAdapter Class
The SqlDataAdapter class represents a bridge between the dataset and the SQL Server database. It includes the Select, Insert, Delete and Update commands for loading and updating the data.

The SqlDataReader Class
The SqlDataReader class creates a data reader to be used with SQL Server.

DataReaders

A DataReader is a lightweight object that provides read-only, forward-only data in a very fast and efficient way. Using a DataReader is efficient than using a DataAdapter but it is limited. Data access with DataReader is
read-only, meaning, we cannot make any changes (update) to data and forward-only, which means we cannot go back to the previous record which was accessed. A DataReader requires the exclusive use of an active connection for the entire time it is in existence. We instantiate a DataReader by making a call to a Command object'sExecuteReader command. When the DataReader is first returned it is positioned before the first record of the result set. To make the first record available we need to call the Read method. If a record is available, the Read method moves the DataReader to next record and returns True. If a record is not available the Read method returns False. We use a While Loop to iterate through the records with the Read method.

Sample Code

Code to Retrieve Data using Select Command

The following code displays data from Discounts table in Pubs sample database.

```
Imports System.Data.SqlClient
Public Class Form1 Inherits System.Windows.Forms.Form
Dim myConnection As SqlConnection
Dim myCommand As SqlCommand
Dim dr As New SqlDataReader()
'declaring the objects

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs)_
Handles MyBase.Load
```

```
myConnection = New
SqlConnection("server=localhost;uid=sa;pwd=;database=pubs")
'establishing connection. you need to provide password for sql server
Try
myConnection.Open()
'opening the connection
myCommand = New SqlCommand("Select * from discounts",
myConnection)
'executing the command and assigning it to connection
dr = myCommand.ExecuteReader()
While dr.Read()
'reading from the datareader
MessageBox.Show("discounttype" & dr(0).ToString())
MessageBox.Show("stor_id" & dr(1).ToString())
MessageBox.Show("lowqty" & dr(2).ToString())
MessageBox.Show("highqty" & dr(3).ToString())
MessageBox.Show("discount" & dr(4).ToString())
'displaying the data from the table
End While
dr.Close()
myConnection.Close()
Catch e As Exception
End Try
End Sub

End Class
```

The above code displays records from discounts table in MessageBoxes.

Retrieving records with a Console Application

```
Imports System.Data.SqlClient
Imports System.Console
Module Module1

Dim myConnection As SqlConnection
Dim myCommand As SqlCommand
Dim dr As SqlDataReader

Sub Main()
Try
myConnection = New
SqlConnection("server=localhost;uid=sa;pwd=;database=pubs")
'you need to provide password for sql server
myConnection.Open()
myCommand = New SqlCommand("Select * from discounts",
myConnection)
dr = myCommand.ExecuteReader
Do
```

```
While dr.Read()
WriteLine(dr(0))
WriteLine(dr(1))
WriteLine(dr(2))
WriteLine(dr(3))
WriteLine(dr(4))
' writing to console
End While
Loop While dr.NextResult()
Catch
End Try
dr.Close()
myConnection.Close()
End Sub

End Module
```

Using DataReaders, SQL Server

Inserting Records

The following code inserts a Record into the Jobs table in Pubs sample database. Drag a button onto the form and place the following code.

```
Imports System.Data.SqlClient
Public Class Form2 Inherits System.Windows.Forms.Form
Dim myConnection As SqlConnection
Dim myCommand As SqlCommand
Dim ra as Integer
'integer holds the number of records inserted
Private Sub Form2_Load(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles MyBase.Load
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
myConnection = New
SqlConnection("server=localhost;uid=sa;pwd=;database=pubs")
'you need to provide password for sql server
myConnection.Open()
myCommand = New SqlCommand("Insert into Jobs values 12,'IT
Manager',100,300,_
myConnection)
ra=myCommand.ExecuteNonQuery()
MessageBox.Show("New Row Inserted" & ra)
myConnection.Close()
End Sub
End Class
```

Deleting a Record

We will use Authors table in Pubs sample database to work with this code. Drag a button onto the form and place the following code.

```
Imports System.Data.SqlClient
Public Class Form3 Inherits System.Windows.Forms.Form
Dim myConnection As SqlConnection
Dim myCommand As SqlCommand
Dim ra as Integer
Private Sub Form3_Load(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles MyBase.Load
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
myConnection = New
SqlConnection("server=localhost;uid=sa;pwd=;database=pubs")
'you need to provide password for sql server
myConnection.Open()
myCommand = New SqlCommand("Delete from Authors where
city='Oakland'",_
myConnection)
'since no value is returned we use ExecuteNonQuery
ra=myCommand.ExecuteNonQuery()
MessageBox.Show("Records affected" & ra)
myConnection.Close()
End Sub
End Class
```

Updating Records

We will update a row in Authors table. Drag a button onto the form and place the following code.

```
Imports System.Data.SqlClient
Public Class Form4 Inherits System.Windows.Forms.Form
Dim myConnection As SqlConnection
Dim myCommand As SqlCommand
Dim ra as Integer
Private Sub Form4_Load(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles MyBase.Load
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
myConnection = New
SqlConnection("server=localhost;uid=sa;pwd=;database=pubs")
'you need to provide password for sql server
myConnection.Open()
myCommand = New SqlCommand("Update Authors Set city='Oakland'
```

```
where city=_
'San Jose' ",myConnection)
ra=myCommand.ExecuteNonQuery()
MessageBox.Show("Records affected" & ra)
myConnection.Close()
End Sub
End Class
```

Using OleDb Provider

The Objects of the OleDb provider with which we work are:

The OleDbConnection Class
The OleDbConnection class represents a connection to OleDb data source. OleDb connections are used to connect to most databases.

The OleDbCommand Class
The OleDbCommand class represents a SQL statement or stored procedure that is executed in a database by an OLEDB provider.

The OleDbDataAdapter Class
The OleDbDataAdapter class acts as a middleman between the datasets and OleDb data source. We use the Select, Insert, Delete and Update commands for loading and updating the data.

The OleDbDataReader Class
The OleDbDataReader class creates a data reader for use with an OleDb data provider. It is used to read a row of data from the database. The data is read as forward-only stream which means that data is read sequentially, one row after another not allowing you to choose a row you want or going backwards.

Sample Code

We will work with the sample Emp table in Oracle.

Retrieving Records

```
Imports System.Data.OleDB
Public Class Form1 Inherits System.Windows.Forms.Form
Dim myConnection As OleDbConnection
Dim myCommand As OleDbCommand
Dim dr As New OleDbDataReader()
'declaration
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs)_
Handles MyBase.Load
myConnection = New OleDbConnection_
("Provider=MSDAORA.1;UserID=scott;password=tiger; database=ora")
'MSDORA is the provider when working with Oracle
```

```
Try
myConnection.Open()
'opening the connection
myCommand = New OleDbCommand("Select * from emp",
myConnection)
'executing the command and assigning it to connection
dr = myCommand.ExecuteReader()
While dr.Read()
'reading from the datareader
MessageBox.Show("EmpNo" & dr(0))
MessageBox.Show("EName" & dr(1))
MessageBox.Show("Job" & dr(2))
MessageBox.Show("Mgr" & dr(3))
MessageBox.Show("HireDate" & dr(4))
'displaying data from the table
End While
dr.Close()
myConnection.Close()
Catch e As Exception
End Try
End Sub
End Class
```

The above code displays first 5 columns from the Emp table in Oracle.

Inserting Records

Drag a Button from the toolbox onto the Form. When this Button is clicked the values specified in code will be inserted into the Emp table.

```
Imports System.Data.OleDb
Public Class Form2 Inherits System.Windows.Forms.Form
Dim myConnection As OleDbConnection
Dim myCommand As OleDbCommand
Dim ra as Integer
'integer holds the number of records inserted
Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles MyBase.Load
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click
myConnection = New
OleDbConnection(""Provider=MSDAORA.1;User_   ID=scott;password=tiger;database=ora"
)
Try
myConnection.Open()
myCommand = New OleDbCommand("Insert into emp values 12,'Ben','Salesman',300,_
12-10-2001,3000,500,10 ", myConnection)
'emp table has 8 columns. You can work only with the columns you want
```

```
ra=myCommand.ExecuteNonQuery()
MessageBox.Show("Records Inserted" & ra)
myConnection.Close()
Catch
End Try
End Sub
End Class
```

Deleting Records

Drag a Button on a new form and paste the following code.

```
Imports System.Data.OleDb
Public Class Form3 Inherits System.Windows.Forms.Form
Dim myConnection As OleDbConnection
Dim myCommand As OleDbCommand
Dim ra as Integer
Private Sub Form3_Load(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles MyBase.Load
End Sub
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
Try
myConnection = New OleDbConnection(""Provider=MSDAORA.1;User_
ID=scott;password=tiger;database=ora")
myConnection.Open()
myCommand = New OleDbCommand("Delete from emp where
DeptNo=790220",_
myConnection)
ra=myCommand.ExecuteNonQuery()
MessageBox.Show("Records Deleted" & ra)
myConnection.Close()
Catch
End Try
End Sub
End Class
```

Updating Records

Drag a Button on a new form and paste the following code.

```
Imports System.Data.OleDb
Public Class Form4 Inherits System.Windows.Forms.Form
Dim myConnection As OleDbConnection
Dim myCommand As OleDbCommand
Dim ra as Integer
Private Sub Form4_Load(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles MyBase.Load
End Sub
```

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
Try
myConnection = New OleDbConnection(""Provider=MSDAORA.1;User_
ID=scott;password=tiger;database=ora")
myConnection.Open()
myCommand = New OleDbCommand("Update emp Set DeptNo=65
where DeptNo=793410",_ myConnection)
ra=myCommand.ExecuteNonQuery()
MessageBox.Show("Records Updated" & ra)
myConnection.Close()
Catch
End Try
End Sub
End Class
```

Data Access using MSAccess

To work with Microsoft Access we use the OleDb data Provider.

Sample Code

Create a database named Emp in Microsoft Access in the C: drive of your machine. In the Emp database create a table, Table1 with EmpNo, EName and Department as columns, insert some values in the table and close it. Open Visual Studio .NET, on a new form drag three TextBoxes and a Button. The following code will assume that TextBox1 is for EmpNo, TextBox2 is for EName and TextBox3 is for Department. Our intention is to retrieve data from Table1 in the Emp Database and display the values in these TextBoxes without binding when the Button is clicked.

Code for retrieving records

```
Imports System.Data.OleDb
Public Class Form1 Inherits System.Windows.Forms.Form
Dim cn As OleDbConnection
Dim cmd As OleDbCommand
Dim dr As OleDbDataReader
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e as _
System.EventArgs) Handles MyBase.Load
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
Try
cn = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;_
Data Source=C:\emp.mdb;")
'provider to be used when working with access database
cn.Open()
cmd = New OleDbCommand("select * from table1", cn)
```

```
dr = cmd.ExecuteReader
While dr.Read()
TextBox1.Text = dr(0)
TextBox2.Text = dr(1)
TextBox3.Text = dr(2)
' loading data into TextBoxes by column index
End While
Catch
End Try
dr.Close()
cn.Close()
End Sub
End Class
```

When you run the code and click the Button, records from Table1 of the Emp database will be displayed in the TextBoxes.

Retrieving records with a Console Application

```
Imports System.Data.OleDb
Imports System.Console
Module Module1

Dim cn As OleDbConnection
Dim cmd As OleDbCommand
Dim dr As OleDbDataReader

Sub Main()
Try
cn = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\emp.mdb;_
Persist Security Info=False")
cn.Open()
cmd = New OleDbCommand("select * from table1", cn)
dr = cmd.ExecuteReader
While dr.Read()
WriteLine(dr(0))
WriteLine(dr(1))
WriteLine(dr(2))
'writing to console
End While
Catch
End Try
dr.Close()
cn.Close()
End Sub

End Module
```

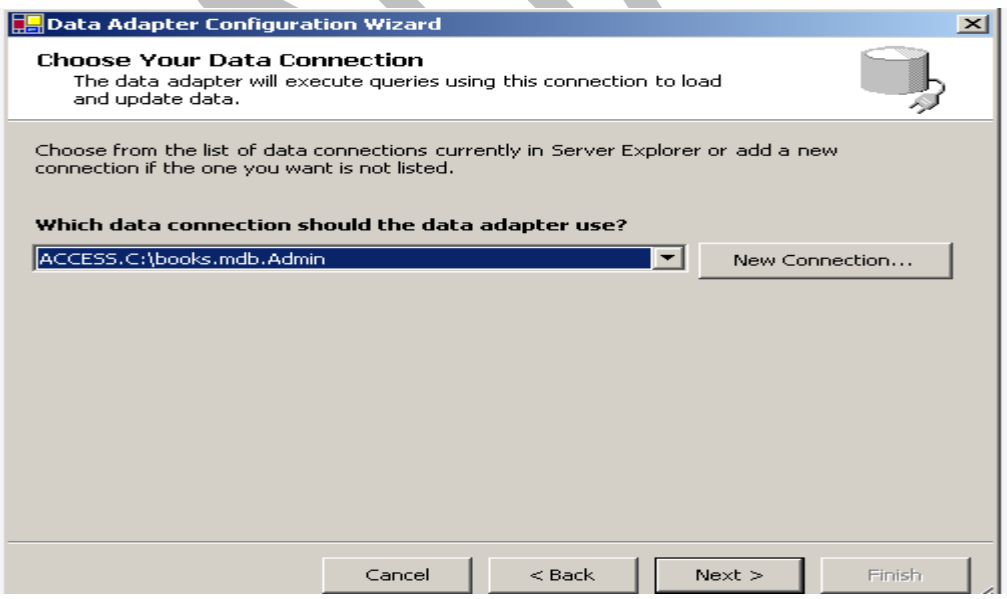
Code for Inserting a Record

```
Imports System.Data.OleDb
Public Class Form2 Inherits System.Windows.Forms.Form
Dim cn As OleDbConnection
Dim cmd As OleDbCommand
Dim dr As OleDbDataReader
Dim icount As Integer
Dim str As String

Private Sub Form2_Load(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles MyBase.Load

End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button2.Click
Try
cn = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\emp.mdb;")
cn.Open()
str = "insert into table1 values(" & CInt(TextBox1.Text) & ",'" &
TextBox2.Text & "','" &_
TextBox3.Text & "')"
'string stores the command and CInt is used to convert number to string
cmd = New OleDbCommand(str, cn)
icount = cmd.ExecuteNonQuery
MessageBox.Show(icount)
'displays number of records inserted
Catch
End Try
cn.Close()
End Sub
End Class
```

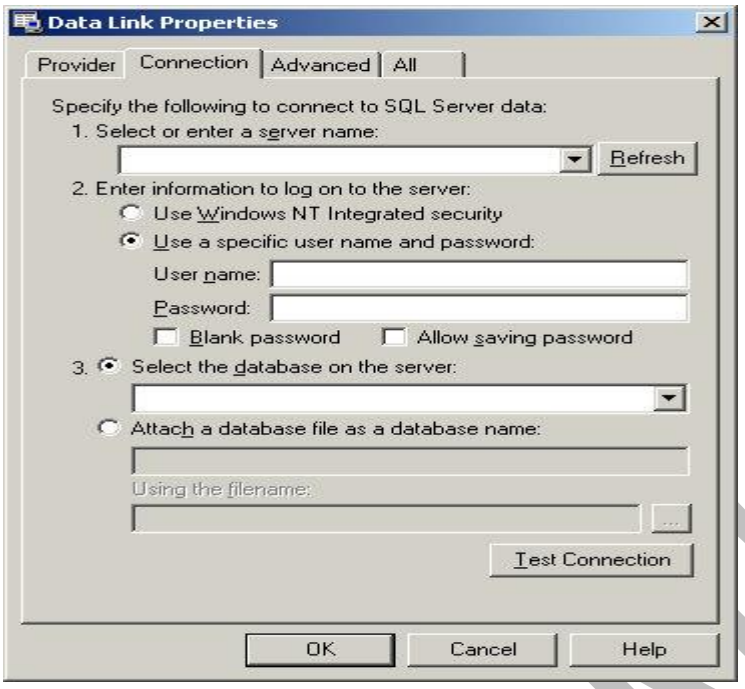
Data Adapter Configuration Wizard

In this section we will create our own data adapter, a built-in feature that comes with Visual Basic .NET and work with it. We will create our own table and access data from the newly created table. To start, create a new database in Access, name it as Books, create a table, Table1 with some columns in it and make sure the database is in the C: drive of your machine. To start creating your own DataAdapter, open a blank form and add a button (Button1) and a DataGrid control to it from the toolbox. Our intention here is to display the table or some columns from the table which we created in Access in the DataGrid control when Button1 is clicked. To display that, click on the Data tab in the toolbox and double-click OleDbDataAdapter object. We are using OleDbDataAdapter here as we are working with an OleDb data source. After you select OleDbDataAdapter from the data tab in the toolbox it gets added to the component tray beneath the form and opens the Data Adapter Configuration Wizard dialogue which looks like the image below.

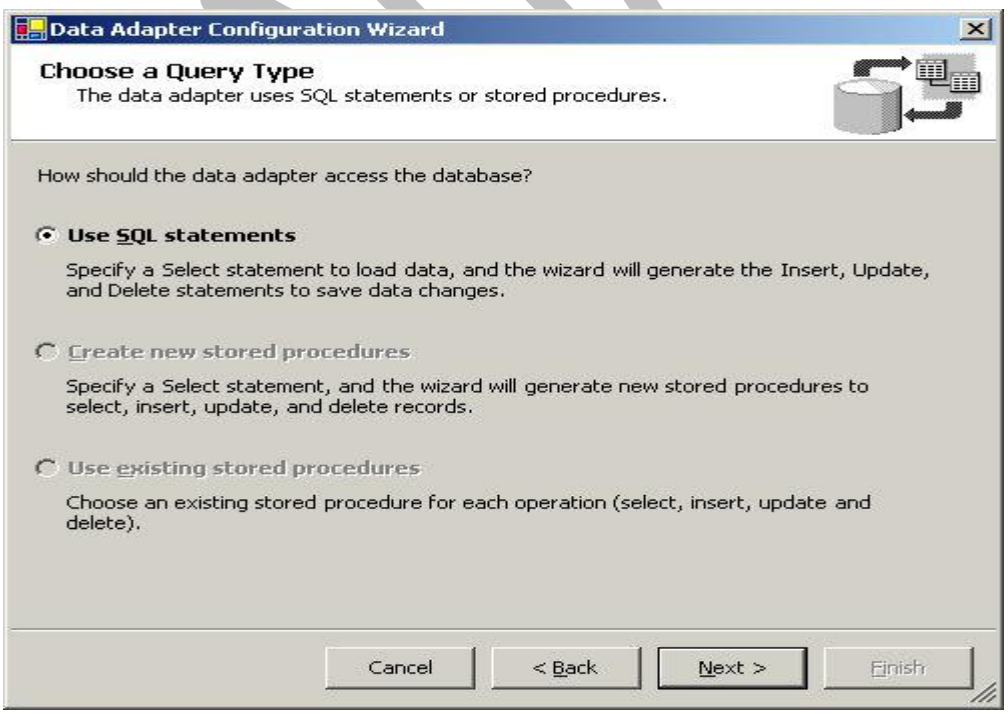
The DataAdapter Configuration wizard let's you customize your data adapter as you want, example, displaying the whole table or displaying selected columns from the tableand so on. Click the next button in the Data Adapter Configuration wizard to select the data connection you want to use. The dialogue that opens look like the image below. It allows you to choose the data connection.



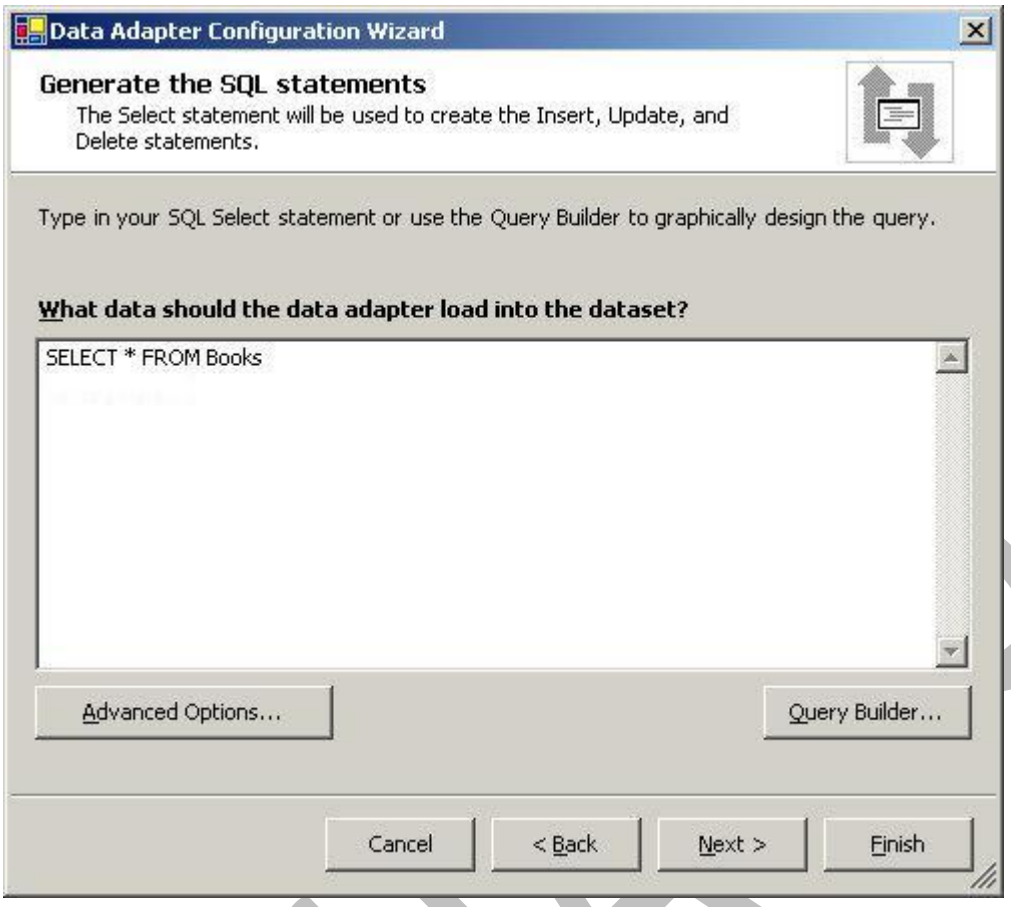
Since we are working with the table we created, click the "New Connection" button in this dialogue box which opens the Data Link properties window. The Data Link Properties window looks like the image below.



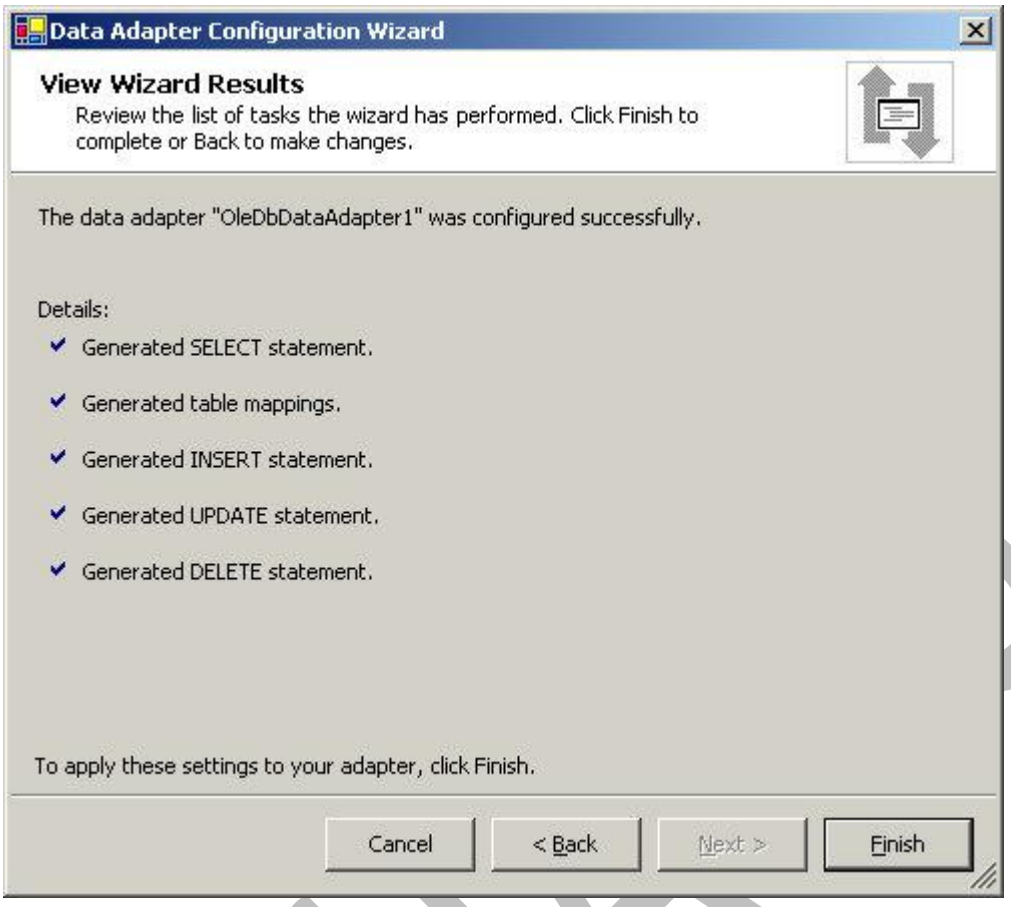
In the Data Link properties window, select the Provider tab and select "Microsoft Jet 4.0 OLE DB Provider" from the list of available providers. After selecting the provider, select the Connection tab. Click on the ellipse where it says "Select or enter a database name" and browse for the database on the local drive. Since we are workingwith our own database (Books.mdb) located on the C: drive select that. Click on the "Test Connection" button to test the connection and if the connection succeeds click OK. Clicking OK display a dialogue box like the image below.



It's at this stage we will generate the SQL Statements to be used with this data adapter. Click next on this dialog box which takes you to another dialogue box like the image below.



It's here we build our SQL Queries. We can display the entire table in the DataGrid or just some columns from the table. To display data, click on the Query Builder button on this dialog box to build your queries. Once you click that button a new dialog box opens up with a list that displays all the tables in the database with which we areworking. In this case it displays only one table as we created only one table in the Books database. Select Table1 and click Add. Table1 is added to the Query Builder window. You can select entire table to be displayed in the DataGrid or just some columns. To display entire table in the DataGrid select the checkbox named "All Columns" in the small dialog named "Table1" which automatically builds the SQL statement for us. If you want to display specific columns from the table in the DataGrid check on the columns you want to display. Once you finish with your selection, click next. The dialogue box that opens when you click next looks like the image below.



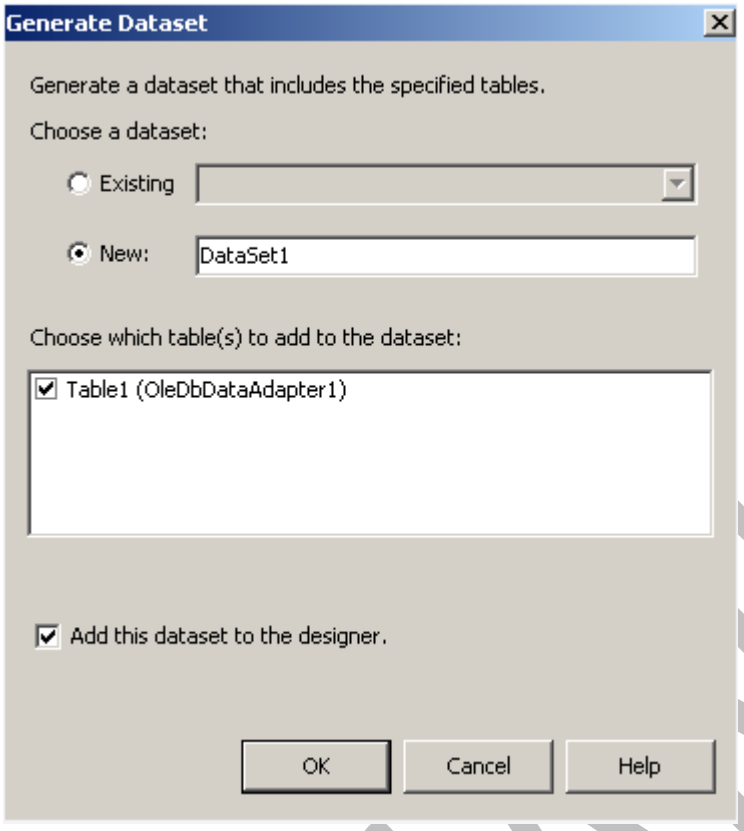
This dialogue lists the configuration of the data adapter and the results. Click finish to close the Data Adapter Configuration wizard.

That creates the data adapter, DataAdapter1, we need. Next step is to create a DataSet and connect this DataSet to the DataGrid using the DataSource and DataMemberproperties.

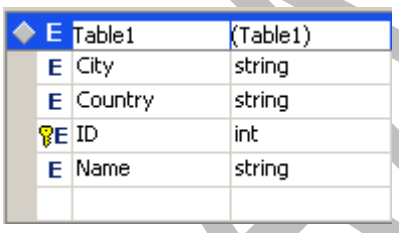
Data Adapter Configuration Wizard

Generating DataSet

To create a DataSet, select Data->Generate DataSet from the main menu. From the dialogue that opens, select new and check the checkbox for Table1 and also check thecheckbox where it says "add this dataset to the designer" and click OK. Once you click OK you can see the DataSet, DataSet11 being added to the component tray. The image below displays generate dataset dialogue.



You can also see the XML schema file named DataSet1.xsd that is generated to define the DataSet. You can double-click this file in Solution Explorer window if you want to see the schema code. This file is generated because ADO .NET transfers data in XML format. The image below displays that.

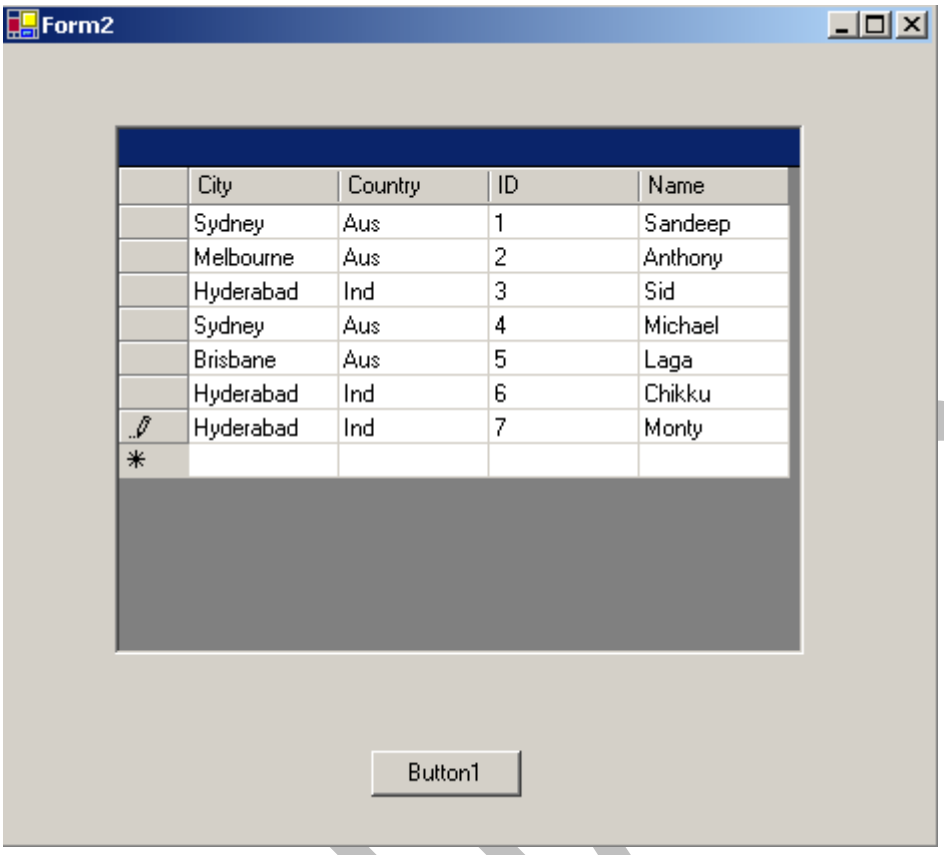


Now, in the properties window for DataGrid, select the DataSource property. The DataSource displays the DataSet which we generated. Select DataSet11 from the list and in the DataMember property select Tabel1. The following lines of code will fill the DataGrid with data from the database (Books.mdb) when Button is clicked.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click
DataSet11.Clear()
OleDbDataAdapter1.Fill(DataSet11)
End Sub
```

When you run the application and click the button, entire table or the columns you selected from the table will be displayed in the DataGrid. This is also a finest example of Complex

Data Binding where we bound an entire data table to the data grid. Instead of displaying one data item at a time the data grid displayed entire data table at once.The image below displays an entire table in data grid.



Committing changes in a DataSet

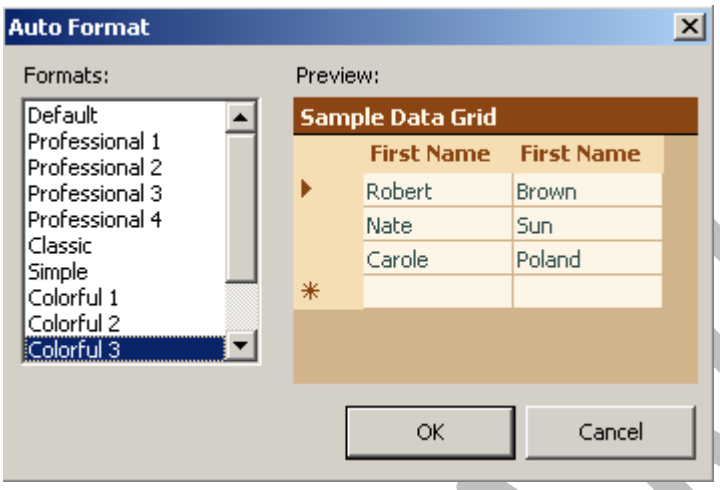
As we make changes to records in a dataset by updating, inserting, and deleting records, the dataset maintains original and current versions of the records. Recall that the DataSet is an in-memory representation of data. All changes we do to records displayed in the data grid above are not committed back to the database. To commit all changes we do to records in dataset we need to call its AcceptChanges method. To do that, add one more button to the form and paste the following code in it's click event.

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button2.Click
OleDbDataAdapter1.Update(DataSet11, "table1")
DataSet11.Table1.AcceptChanges()
End Sub
```

You can also call the Update method of the DataAdapter to commit changes back to the database. It looks like this in code: **OleDbDataAdapter1.Update()**.

Customizing the DataGrid Control

You can also customize the appearance of a data grid. You can customize the data grid by setting it's properties or by selecting the auto format dialog. To open the autoformat dialog, right-click on the data grid and select Auto Format. You can also select it by clicking the Auto Format link found towards the bottom of the data grid properties window. The auto format dialog that opens looks like the image below.



As you can see from the auto format dialog image above you can set the style for the data grid from predefined formats. This dialog lets you select from a number of predefined styles for the data grid, setting header color, border color and so on.

Some of the common appearance and display properties of the data grid control as seen in the properties window are as follows.

Display Properties

CaptionVisible: Determines whether or not the caption area is displayed
ColumnHeadersVisible: Determines whether or not the column headers are displayed
RowHeadersVisible: Determines whether or not the row headers are visible

Appearance Properties

BorderStyle: Gets/Sets the appearance of the data grid border
CaptionFont: Gets/Sets the font that's used to display the text in the caption area
Captiontext: Gets/Sets the text that's displayed in the caption area
FlatMode: Determines whether or not the grid has a flat appearance
Font: Gets/Sets the font that's used to display the text in the data grid
GridLineStyle: Determines whether a solid line or no line is displayed between rows in the data grid
HeaderFont: Sets the font that's used to display the text in the row and column headers

Simple Binding

Data Binding

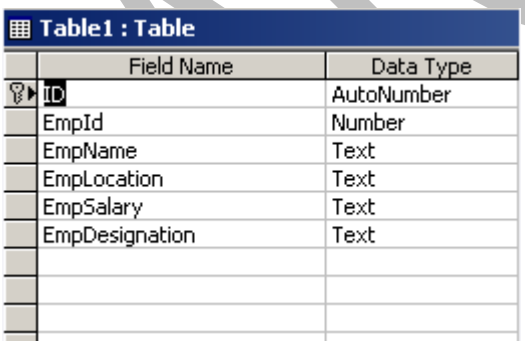
Data Binding is binding controls to data from the database. With data binding we can bind a control to a particular column in a table from the database or we can bind the whole table to the data grid. Data binding provides simple, convenient, and powerful way to create a read/write link between the controls on a form and the data in theirapplication. Windows Forms supports binding data to ADO .NET DataSet, Array, ArrayList, etc. A control can be bound to any collection that supports indexed access to the elements in that collection.

Simple Data Binding

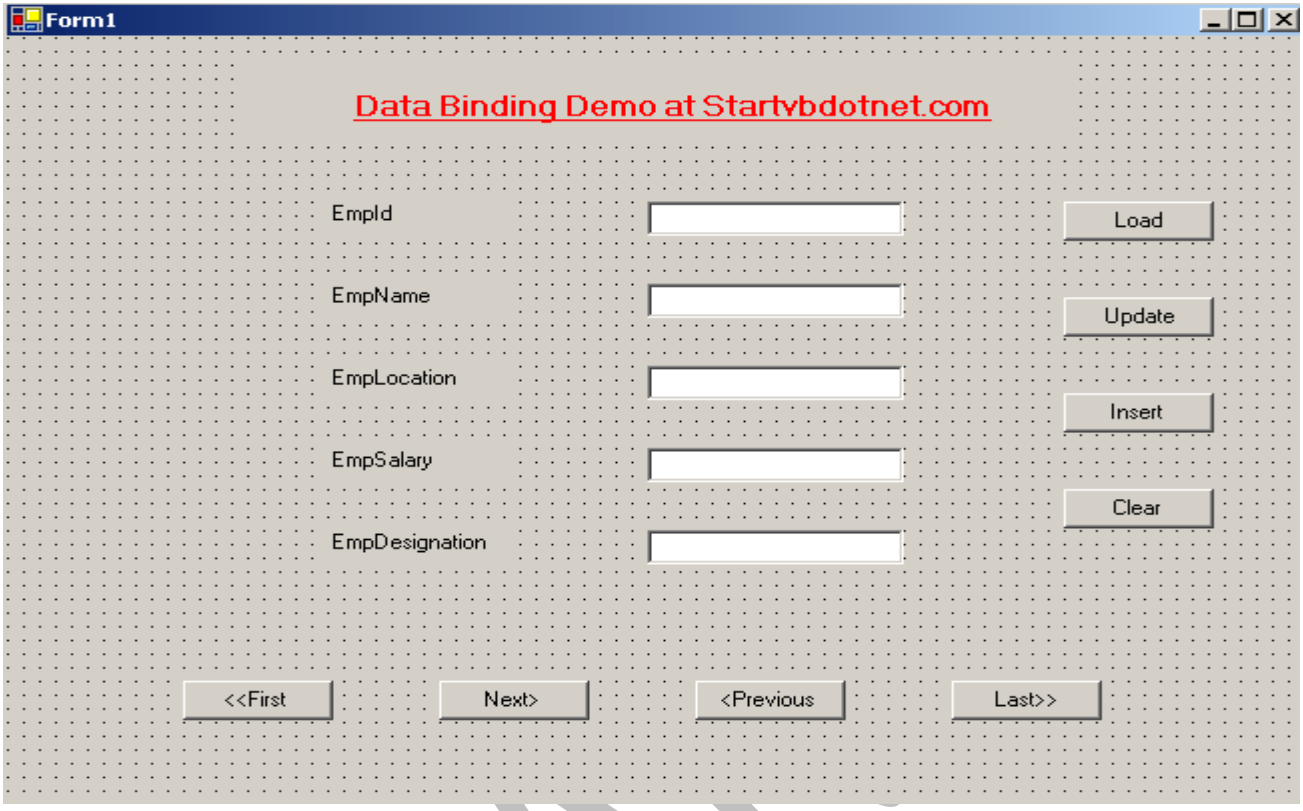
Simple binding allows us to display one data element from a table in a control. Simple binding is managed by use of the Bindings collection on each control. Simple bound controls show only one data element at a time. We have to create our own navigation controls (buttons) to see other data elements. These navigation controls allow us to move from record to record by clicking buttons and the data in the bound controls will be updated automatically. To bind any property of a control, you need to click the ellipse button of the Advanced property under Data Bindings property in the properties window. Clicking on the ellipse opens up Advanced Data Binding Dialog in which we can bind any property of a control to a data source.

Working with Example

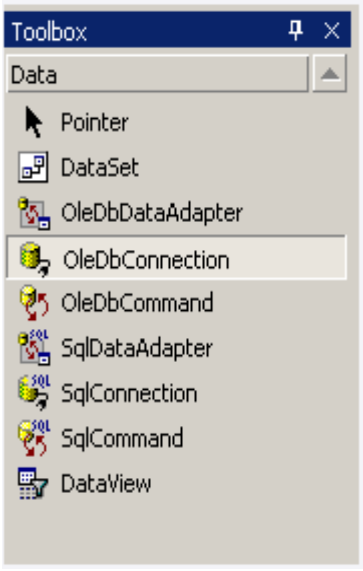
To understand Simple Binding we will create a simple data entry form and work with it. We will create our own table in Access and access data from that table with a Form. To start, open a blank database in MS Access, name it as Emp and save it in the C: drive of your machine. Create a table, Table1 with the following columns, EmpId, EmpName, EmpLocation, EmpSalary and EmpDesignation. The columns and their data types should look like the image below.



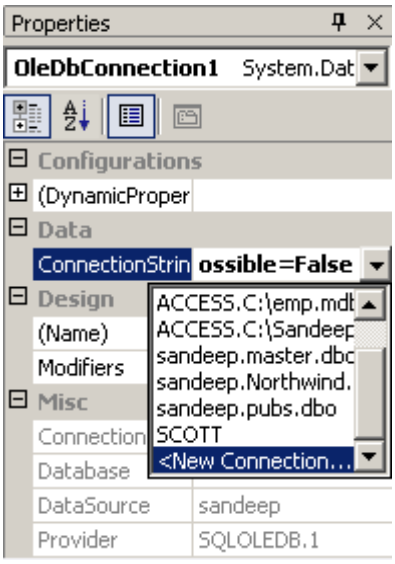
Once you finish creating the table with required columns, enter some values and close it. Get back to Visual Studio, open a new Windows Form and from the toolbox add five TextBoxes, five Labels and eight Buttons. Here, we will bind data from the table we created in Access to TextBoxes. TextBox1 will display EmpId, TextBox2 will display EmpName, TextBox3 will display EmpLocation, TextBox4 will display EmpSalary and TextBox5 will display EmpDesignation. The image below displays the form in design view.



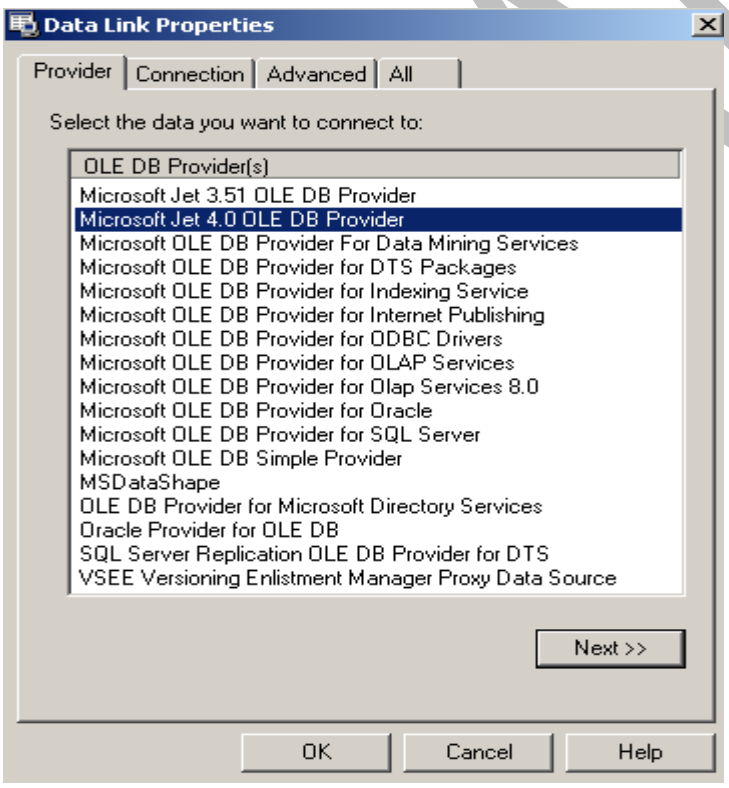
Set the text and name property of all the buttons as shown above in the properties window. Now, in the toolbox, click the Data tab and drag an OleDbConnection object onto the form. The image below displays items from the Data tab.



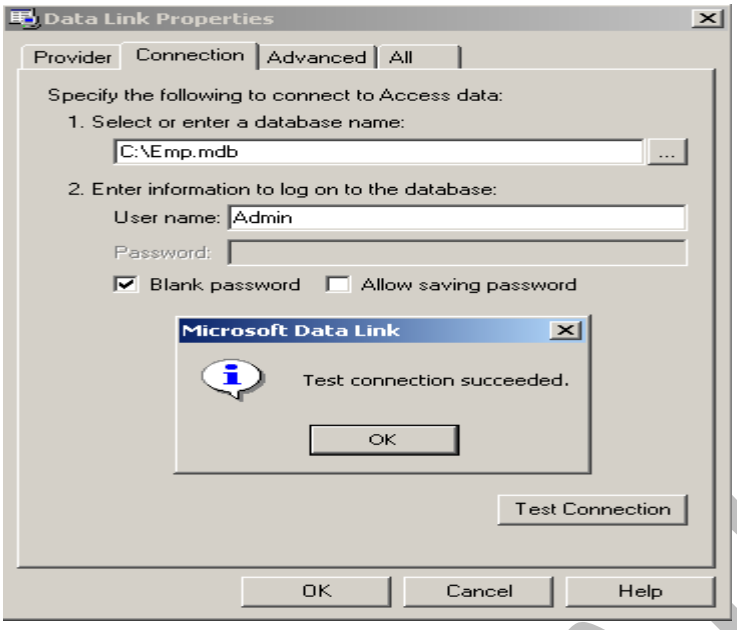
Once the connection object is added to the component tray, open it's properties window to set the connection string. Select ConnectionString property in the properties window, click on the drop-down arrow and select <New Connection...> item. That looks like the image below.



When you select <New Connection...>, it opens the Data Link Properties dialog box. Click on Provider tab in this box and select "Microsoft Jet 4.0 OLE DB Provider". By default it selects provider for SQL Server. After selecting MS Jet, click Next. The image below displays the Data Link properties dialog.



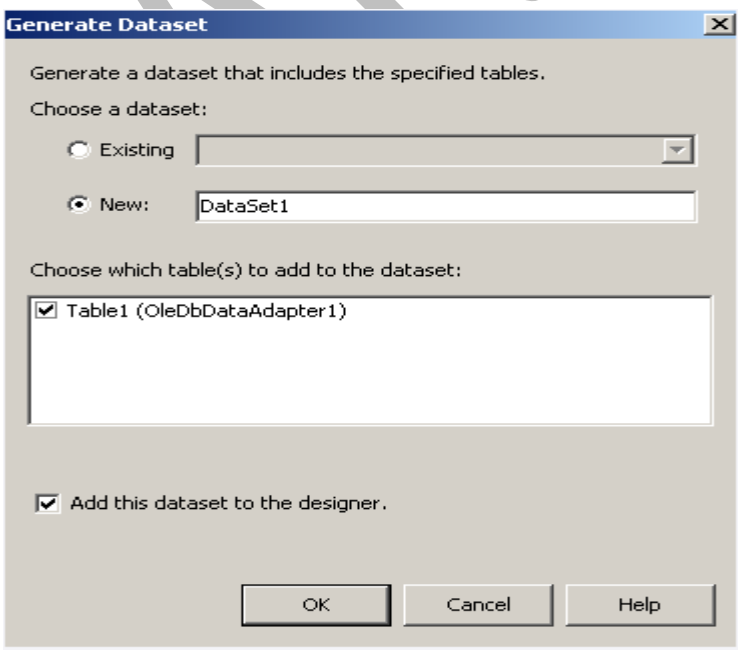
Clicking next takes you to Connection tab. Here, browse for Emp.mdb database in the selection area by clicking the ellipse button and click "Test Connection" button. If connection succeeds, it displays a message box stating that the connection succeeded. The image below displays that.



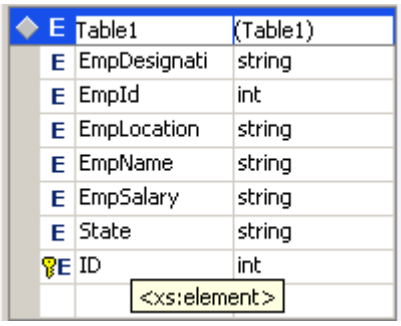
Once you are done with it, click OK. Until this point we created a Connection object that knows how to connect to the database. We still need other objects that will make use of this connection. Get back to the Data tab in toolbox and drag an OleDbDataAdapter tool onto the Form. This adds a new data adapter to the form and automatically starts the Data Adapter Configuration Wizard. You can view configuration of data adapter here. After you finish configuring the data adapter we need to create a Dataset.

Generating DataSet

To generate a DataSet, select Data->Generate DataSet from the main menu. From the dialog that opens, as shown in the image below, select new and check the checkboxfor Table1 and also check the checkbox where it says "add this dataset to the designer" and click OK. Once you click OK, you can see the DataSet, DataSet11 being added to the component tray.

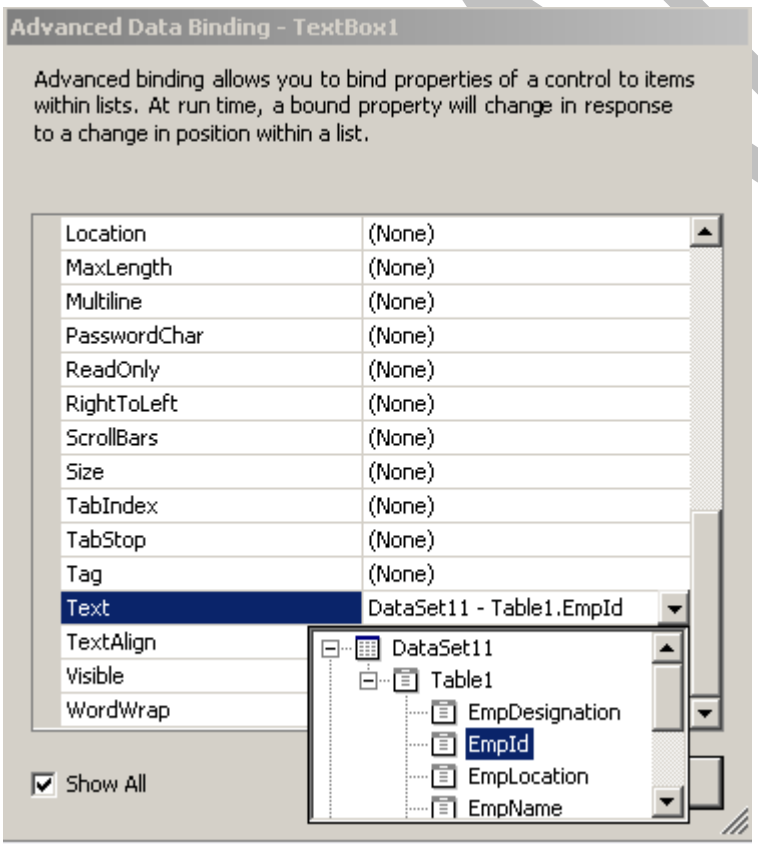


You can also see the XML schema file named DataSet1.xsd that is generated to define the DataSet. You can double-click this file in Solution Explorer window if you want to see the schema code. This file is generated because ADO .NET transfers data in XML format. The image below displays that.



Binding Controls to the DataSet

We are now ready to bind textboxes to this dataset. We will bind the text property of textboxes to data columns in table1. Select TextBox1, open it's properties and underData Bindings, select Advanced and click on the ellipse to open the Advanced Data Binding window for the textbox. It looks like the image below.



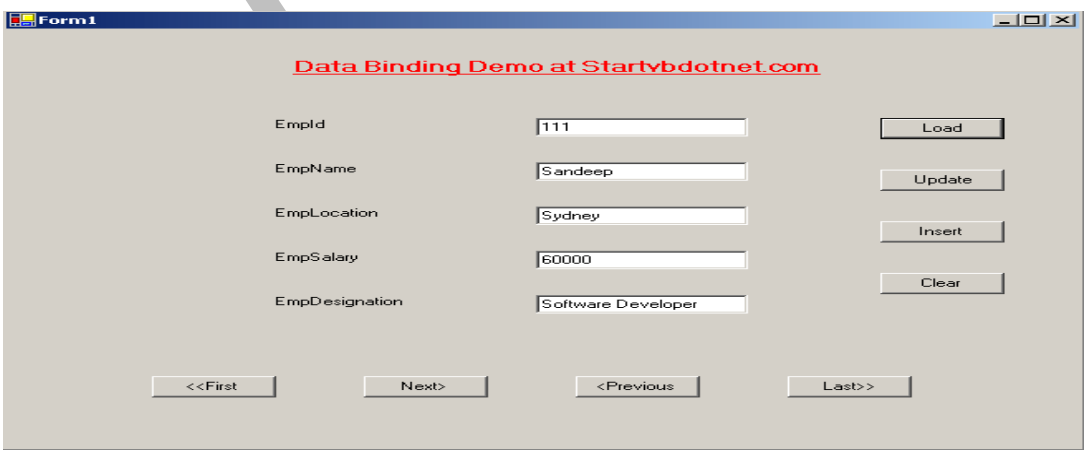
As shown in the image above, select Text property, click on the drop-down arrow, double-click DataSet11 which opens Table1. Double-click Table1 to list all columns in the table. Here, you select the column you want to display in this textbox. Once you finish selecting the column click close. Repeat the process for remaining textboxes until all columns in Table1 are accommodated. Once you finish with all textboxes, we need to write code for buttons to

finish our data-entry form. Switch to code view and place the following code. Make sure you place code correctly for each button.

```
Public Class Form1 Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

#End Region

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs)_
Handles MyBase.Load
'data binding demo

End Sub

Private Sub Load_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs)  Handles Load.Click
OleDbDataAdapter1.Fill(DataSet11)
'filling the dataset and loading records into the textboxes
End Sub

Private Sub Update_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Update.Click
Try
Me.BindingContext(DataSet11, "table1").EndCurrentEdit()
Me.OleDbDataAdapter1.Update(DataSet11)
'ending current editing and updating the dataset
Catch
End Try
End Sub

Private Sub Insert_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Insert.Click
Me.BindingContext(DataSet11, "table1").AddNew()
'adding new record/row to the table
MsgBox("Successfully Inserted")
End Sub

Private Sub Clear_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Clear.Click
TextBox1.Text = " "
TextBox2.Text = " "
TextBox3.Text = " "
TextBox4.Text = " "
TextBox5.Text = " "
'setting all textboxes to null
End Sub

Private Sub First_Click(ByVal sender As System.Object, ByVal e As _
```

```
System.EventArgs) Handles First.Click
Me.BindingContext(DataSet11, "table1").Position = 0
'using forms's BindingContext property's position member and setting it to
0
'displays the first row from table
End Sub

Private Sub NextRec_Click(ByVal sender As System.Object, ByVal e As
_
System.EventArgs) Handles NextRec.Click
Me.BindingContext(DataSet11, "table1").Position =
Me.BindingContext(DataSet11,_
"table1"). Position + 1
'incrementing the position property of the binding context
End Sub

Private Sub Previous_Click(ByVal sender As System.Object, ByVal e As
_
System.EventArgs) Handles Previous.Click
Me.BindingContext(DataSet11, "table1").Position =
Me.BindingContext(DataSet11,_
"table1"). Position - 1
End Sub

Private Sub Last_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Last.Click
Me.BindingContext(DataSet11, "table1").Position =
Me.BindingContext(DataSet11,_
"table1"). Count - 1
'the count property returns the total number of records in the table
End Sub

End Class
```

After finishing with the code, run the application and click Load button. The first row from table1 will be displayed in textboxes. The image below displays that.



You can now move to the last row, next row, previous row, etc or modify your records, insert new records etc. Try experimenting with other controls following the sameprocedure.

CurrencyManager Object

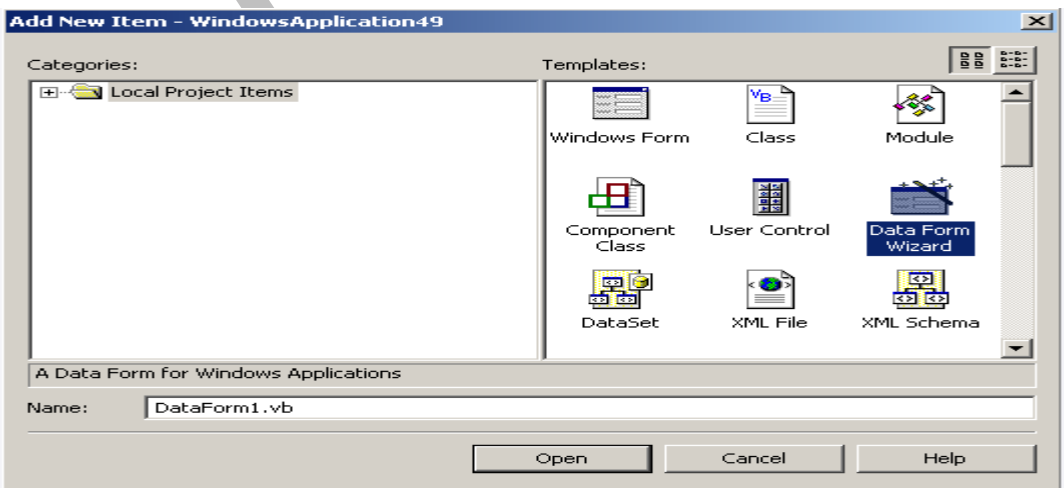
Based on the code above, this is for understanding. Navigation of records and updating of data-bound controls is managed in the data layer as discussed above. Every data source manages navigation with a CurrencyManager object. The CurrencyManager object keeps track of a current record for a particular data source. An application can interact with more than one data source at a given time. Each data source maintains it's own CurrencyManager. Since there can be multiple data sources represented on a single form at any given time, each form manages the CurrencyManager objects associated with those data sources through the central object called the BindingContext. The BindingContext organizes and exposes the CurrencyManager objects associated with each data source. We can use the BindingContext property of each form to manage the position of each record for data source. We access a particular CurrencyManager by supplying the BindingContext property of the CurrencyManager. When navigating records, the current record can be set by setting the Position property for a particular BindingContext.

Simple Binding in Code

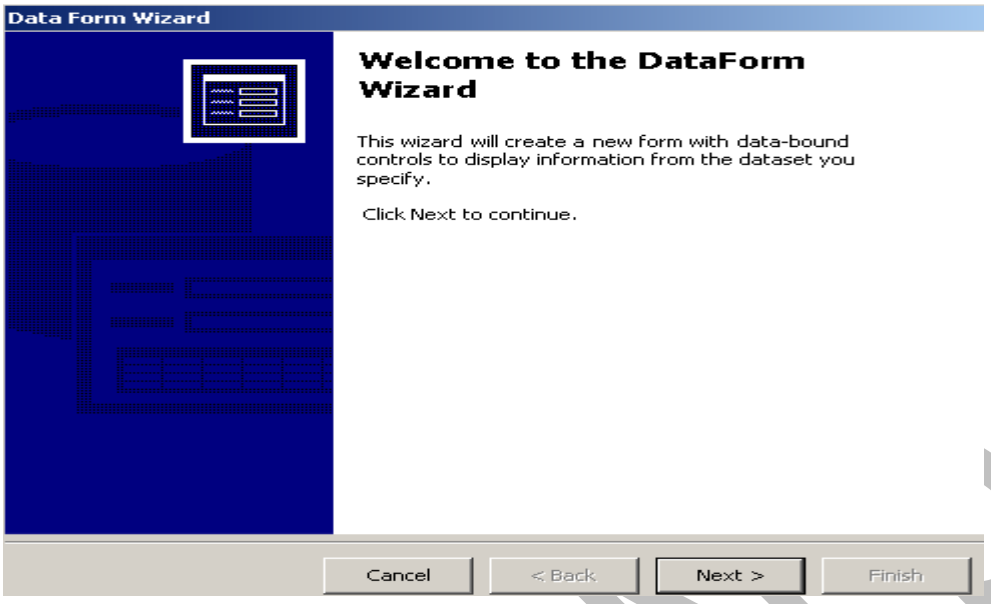
We can also perform simple binding in code using the control's DataBindings property. Say, we want to bind the textbox to the EmpID column in code. We can do that using the collection's Add method by passing this method the property to bind, the data source to use and the specific column we want to bind. The code for that looks like this: **TextBox1.DataBindings.Add("Text", DataSet11, "table1.empid")**.

Data Form Wizard

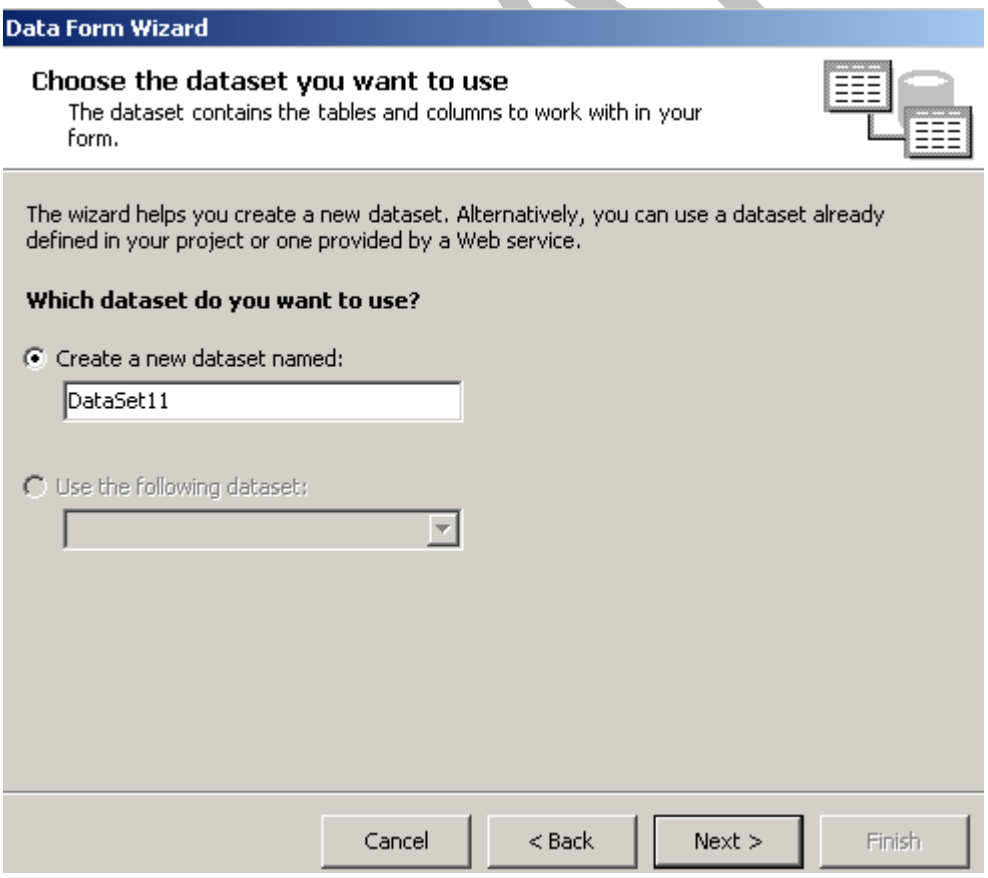
Visual Basic also allows us to work with DataBinding with it's built-in feature "Data Form Wizard". We will have a look at how we can create our own data-entry forms with the Data Form Wizard. A Data Form Wizard is the easiest and fastest way to develop database applications without writing a single line of code. We will create a form and work with Order Details table from the sample Northwind database in SQLServer. To start working, select Project->Add New Item->Data Form Wizard from the main menu. The dialogue box for that looks like the image below.



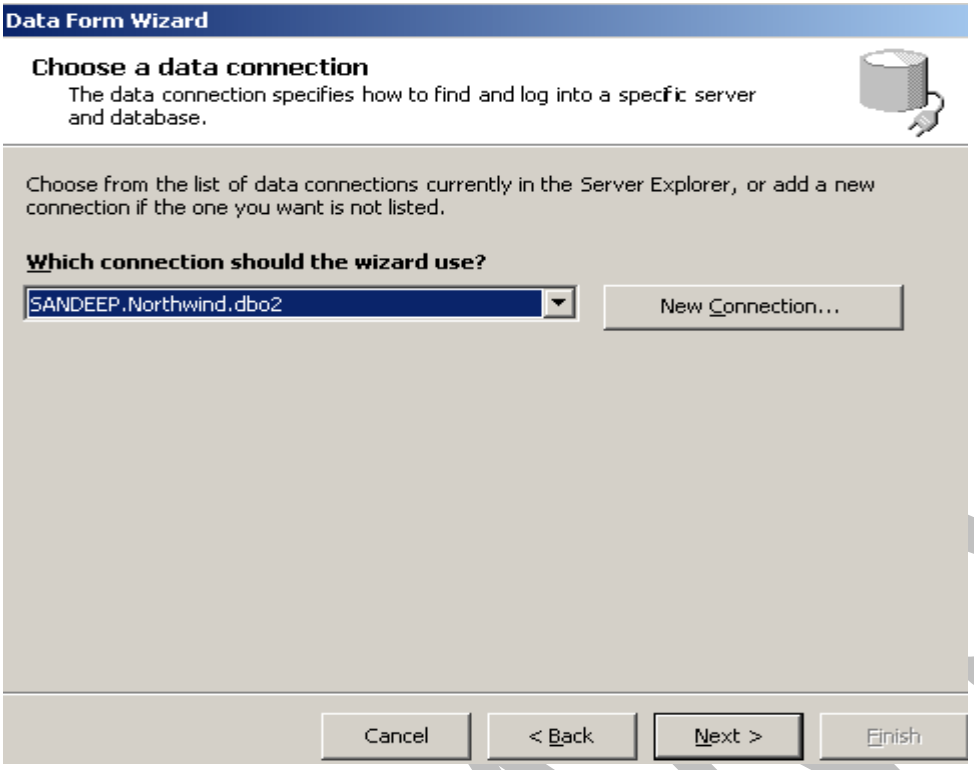
After selecting Data Form Wizard, click Open. The new pane that opens is the DataForm Wizard and it looks like the image below.



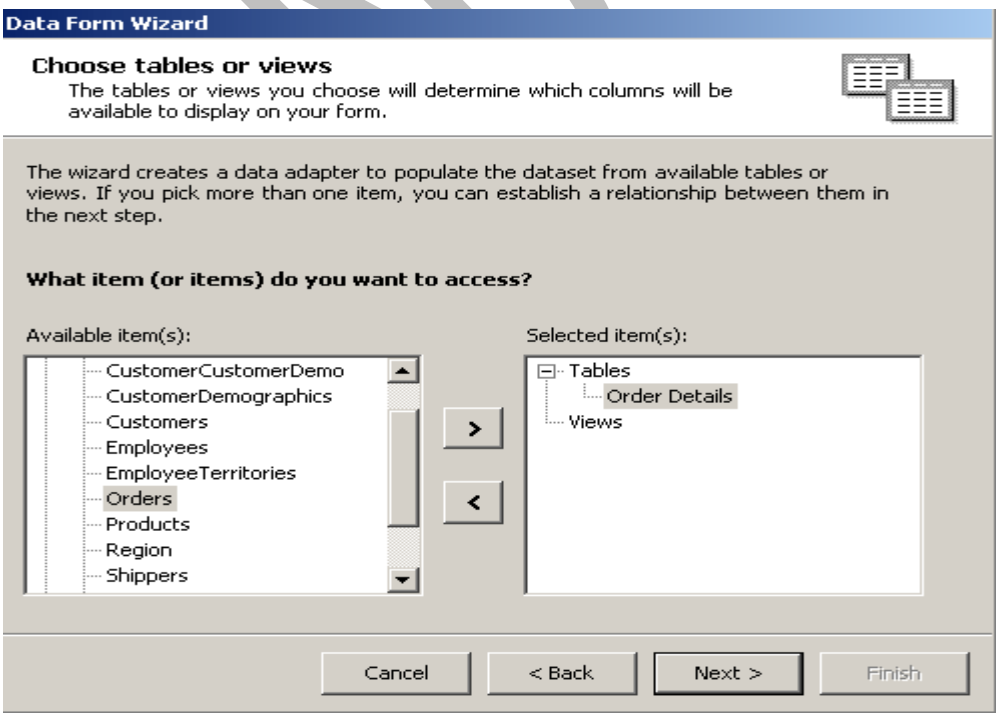
Click Next on this pane. Clicking next takes you to a new pane which looks like the image below. Here you need to specify the name for your DataSet.



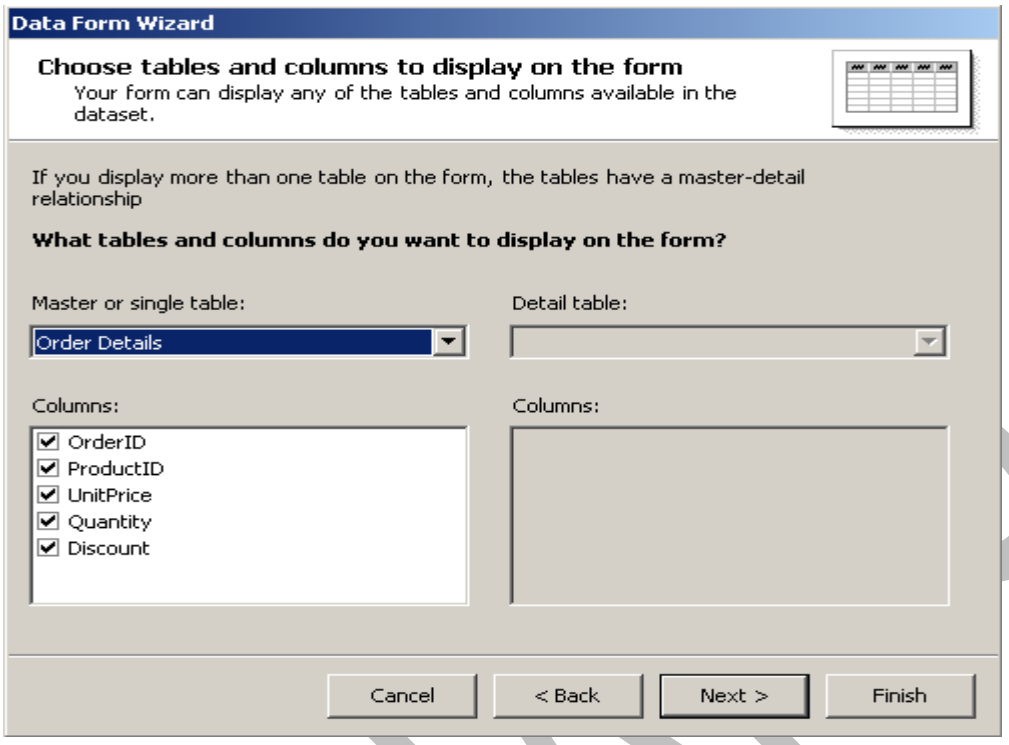
Here, select the radio button that displays "Create a new dataset named", type a name for the DataSet, and click next. Clicking next opens a pane like the image below.



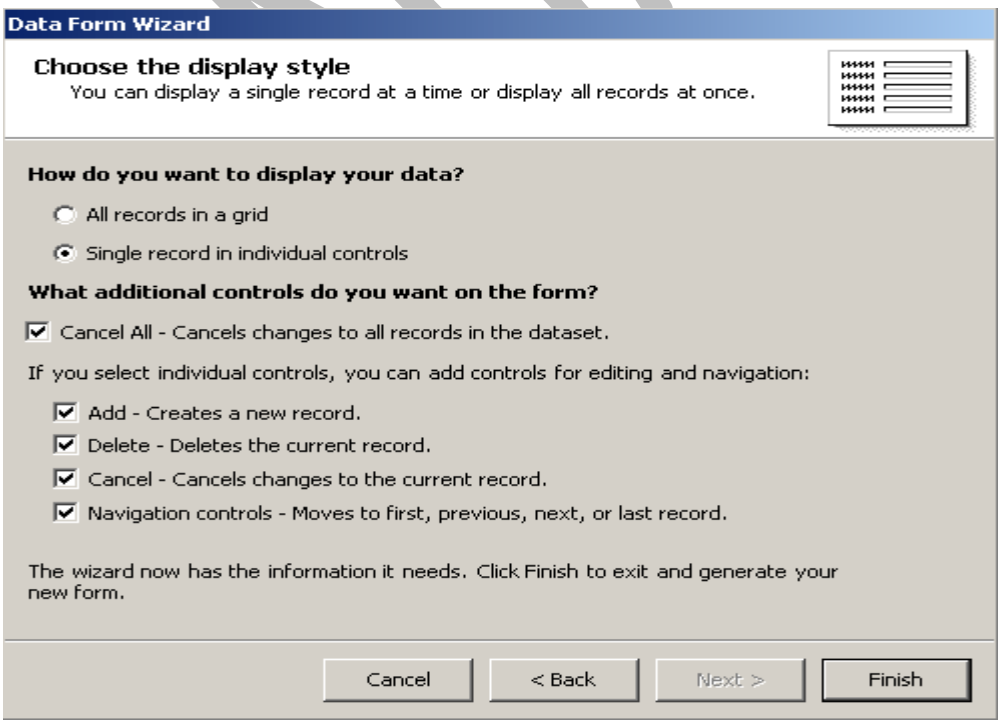
Here we need to establish a connection to the database. Click on the "New Connection" button to open the "Data Link Properties" window. Set a connection to the database in the Data Link properties window. Here, I am using Northwind database. You can use any database you wish to work with. If you already have a connection you can use it. Once you finish with the Connection, click next. Clicking next takes you to a new pane like the image below.



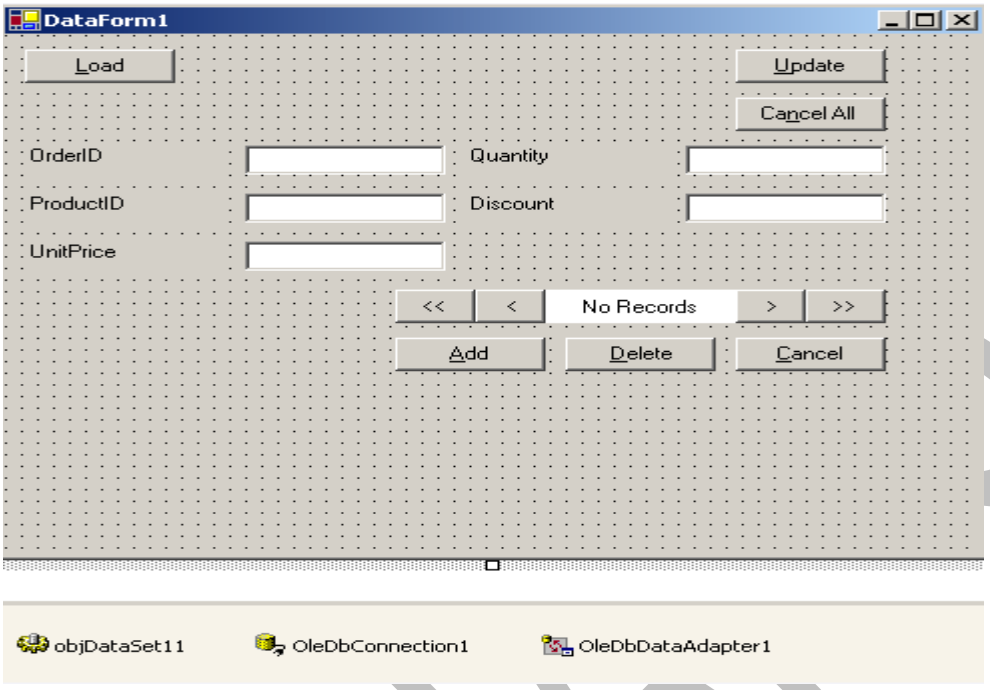
This pane displays all the tables available in Northwind database. Select the table you want to work and add it using the forward arrow button. I am selecting the Order Details table. Click next once you are finished. The next pane looks like the image below.



This pane allows us to display columns from more than one table by establishing a master-detail relationship. Since we are working with one table, click next on this window. Clicking next takes us to a pane like the image below.



This pane allows us to choose the display style for data. By default all the records are set to be displayed in a data grid. Here, select the radio button which says "Single records in individual controls" which will unlock all the checkboxes. Uncheck the check box where it says "Cancel All" and click finish. Clicking finish adds the Data Entry Form with all the required textboxes and navigation controls (buttons). The image below displays that.



Run the form and click the Load button located at the top of the form to load the data into the form. Using the navigation controls you can move to the next record, last record, previous record and the first record. You can also enter/delete/update data into the table. Type text in the textboxes and click the insert button to insert data intothe table or select a record and delete it or make changes to a record and update it by clicking the appropriate buttons. Also open the code behind file to take a look at the code Visual Basic generated for us.

Creating DataTable

Visual Basic allows us to create our own Tables and work with them. The DataTable is an in-memory representation of a block of data. We can create our own tables in code using a DataSet and the types defined in the System.Data.OleDb or System.Data.SqlClient namespaces. The following are the core properties that are used while creating a DataTable.

CaseSensitive: Indicates whether string comparisons in the table are case-sensitive or not.
ChildRelations: Returns the collection of child relations of the DataTable (if any).
Columns: Returns the collection of columns that belong to this table.
Constraints: Gets the constraints maintained by this table.
DataSet: Gets the dataset that contains this table.
DefaultView: Gets a customized view of the table that may include a filtered view or a cursor position.
MinimumCapacity: Gets/Sets the initial number of rows in the table.
ParentRelations: Gets the collection of parent relations for this table.

PrimaryKey: Gets/Sets a primary key for the table.
Rows: Returns the collection of rows that belong to this table.
TableName: Gets/Sets the name of the table.

Creating a DataTable in Code

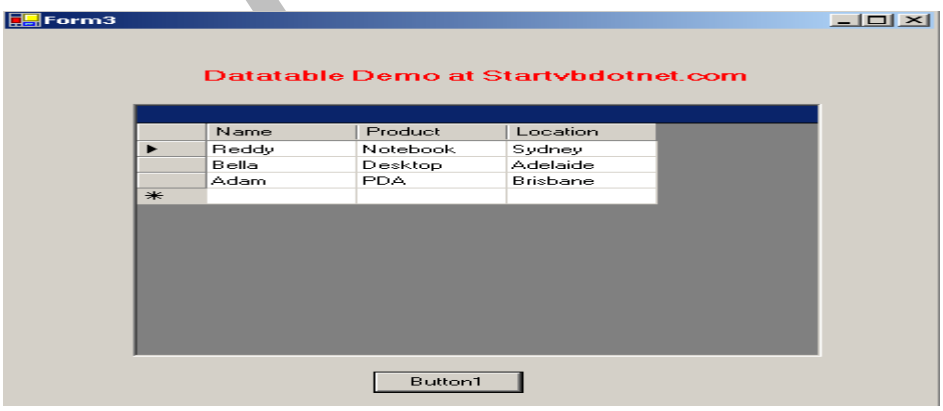
Let's see how we can create a DataTable in code. Open a new form and drag a Button and a DataGrid from the toolbox. We will load the table which we create in code into the DataGrid once the Button is clicked. We will create a DataTable named "**Customers**" , with "**Name**", "**Product**" and "**Location**" as three columns in the table. We will create three rows and three columns for this table. The following code shows how to create a table and load the table into the DataGrid.

```
Public Class Form1 Inherits System.Windows.Forms.Form

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs)_
Handles MyBase.Load

End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
Dim Table1 As DataTable
Table1 = New DataTable("Customers")
'creating a table named Customers
Dim Row1, Row2, Row3 As DataRow
'declaring three rows for the table
Try
Dim Name As DataColumn = New DataColumn("Name")
'declaring a column named Name
Name.DataType = System.Type.GetType("System.String")
'setting the datatype for the column
Table1.Columns.Add(Name)
'adding the column to table
Dim Product As DataColumn = New DataColumn("Product")
Product.DataType = System.Type.GetType("System.String")
Table1.Columns.Add(Product)
Dim Location As DataColumn = New DataColumn("Location")
Location.DataType = System.Type.GetType("System.String")
Table1.Columns.Add(Location)

Row1 = Table1.NewRow()
'declaring a new row
Row1.Item("Name") = "Reddy"
'filling the row with values. Item property is used to set the field value.
Row1.Item("Product") = "Notebook"
'filling the row with values. adding a product
Row1.Item("Location") = "Sydney"
'filling the row with values. adding a location
```

```
Table1.Rows.Add(Row1)
'adding the completed row to the table
Row2 = Table1.NewRow()
Row2.Item("Name") = "Bella"
Row2.Item("Product") = "Desktop"
Row2.Item("Location") = "Adelaide"
Table1.Rows.Add(Row2)
Row3 = Table1.NewRow()
Row3.Item("Name") = "Adam"
Row3.Item("Product") = "PDA"
Row3.Item("Location") = "Brisbane"
Table1.Rows.Add(Row3)
Catch
End Try

Dim ds As New DataSet()
ds = New DataSet()
'creating a dataset
ds.Tables.Add(Table1)
'adding the table to dataset
DataGrid1.SetDataBinding(ds, "Customers")
'binding the table to datagrid
End Sub

End Class
```

When you run the above code and click the Button, a table is created and the created table loads into the DataGrid. You can add any number of rows and columns to the table. I added only three for the purpose of explanation. Other DataType values supported are: System.Boolean, System.Byte, System.Char, System.DateTime,System.Decimal, System .Double, System.Int16, System.Int32, System.Int64, System.SByte, System.Single. You can use any one of the above said data types depending on the type of data you will have in the columns.

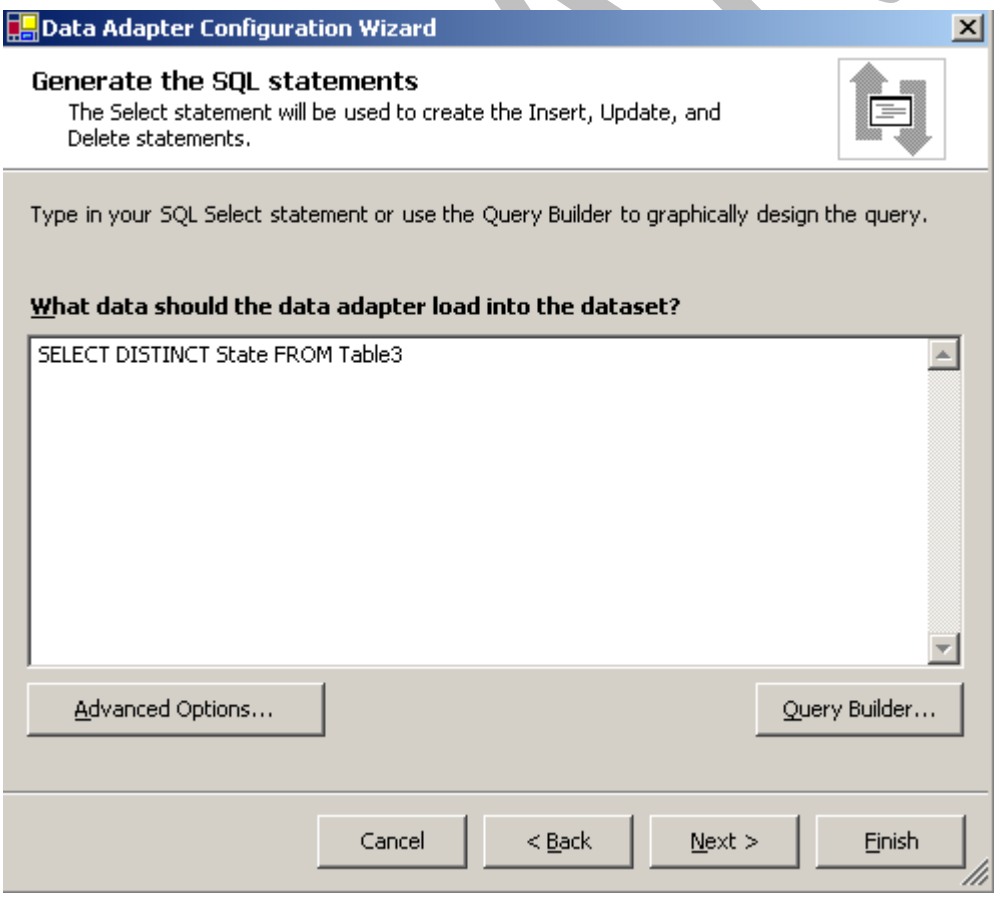
The image below displays output from above code.



Using SQL Parameters

In VB.NET, we have Windows controls that may provide parameters for filtering data from the database. A common scenario might be searching for some data from the Windows form and then populating a DataGrid with the results. Sql Parameters help us to filter data. We will follow the usage of Sql Parameters in VB .NET with a sample application. To start, open a blank Microsoft Access database, name it as Sample. Create a table, Table1 in Sample database with the columns, Business Name, Name, Order ID and State. Enter some values in the table and close it. Make sure you enter atleast 3 different state names when you fill the table with values. The followingsample application will use two data adapters and two datasets to let the user select a state from a ComboBox and display data relating to that state in a data grid. When the form loads, the states from the table load into the dataset bound to the combo box. The user can select a state from the combo box and click the load button on the form to load all the data relating to that state into a second dataset, whose data is displayed in a data gird.

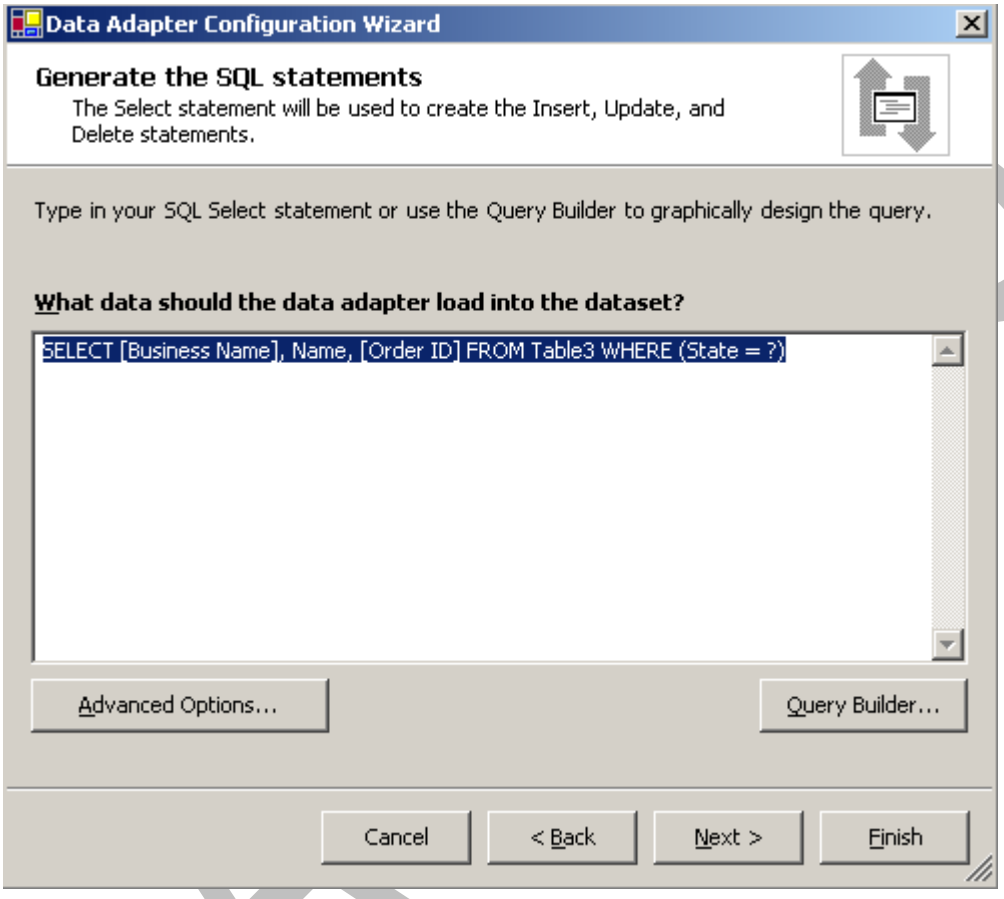
Loading the state data from table1 is fairly simple with the first data adapter. Drag a OleDb Data Adapter from the data tab of the toolbox onto the form. The Data Adapter Configuration Wizard opens and you are required to configure it. You can have a look at the Data Adapter Configuration Wizard here. While configuring the data adapter, the Generate Sql Statements dialog box should be as follows: Select DISTINCT State from Table1 as shown in the image below.



Note that we used the DISTINCT keyword in the SQL Statement above. The distinct keyword is used to load a unique state in the dataset. No state will be displayed more than

once. After you finish configuring the data adapter, you need to generate the dataset. To generate a dataset, DataSet11 select Data->Generate Dataset from the main menu.

After the user selects a state in the ComboBox and clicks the load button the data should load into the data grid. To load data into the data grid you need to use the second data adapter, OleDb DataAdapter2. From the data tab of the toolbox drag an OleDb DataAdapter, OleDb Data Adapter2 onto the form and configure it. While configuring the data adapter the Generate Sql Statements dialog box should be as follows:
Select Business Name, Name, Order ID from Table1 WHERE (state=?) as shown in the image below.



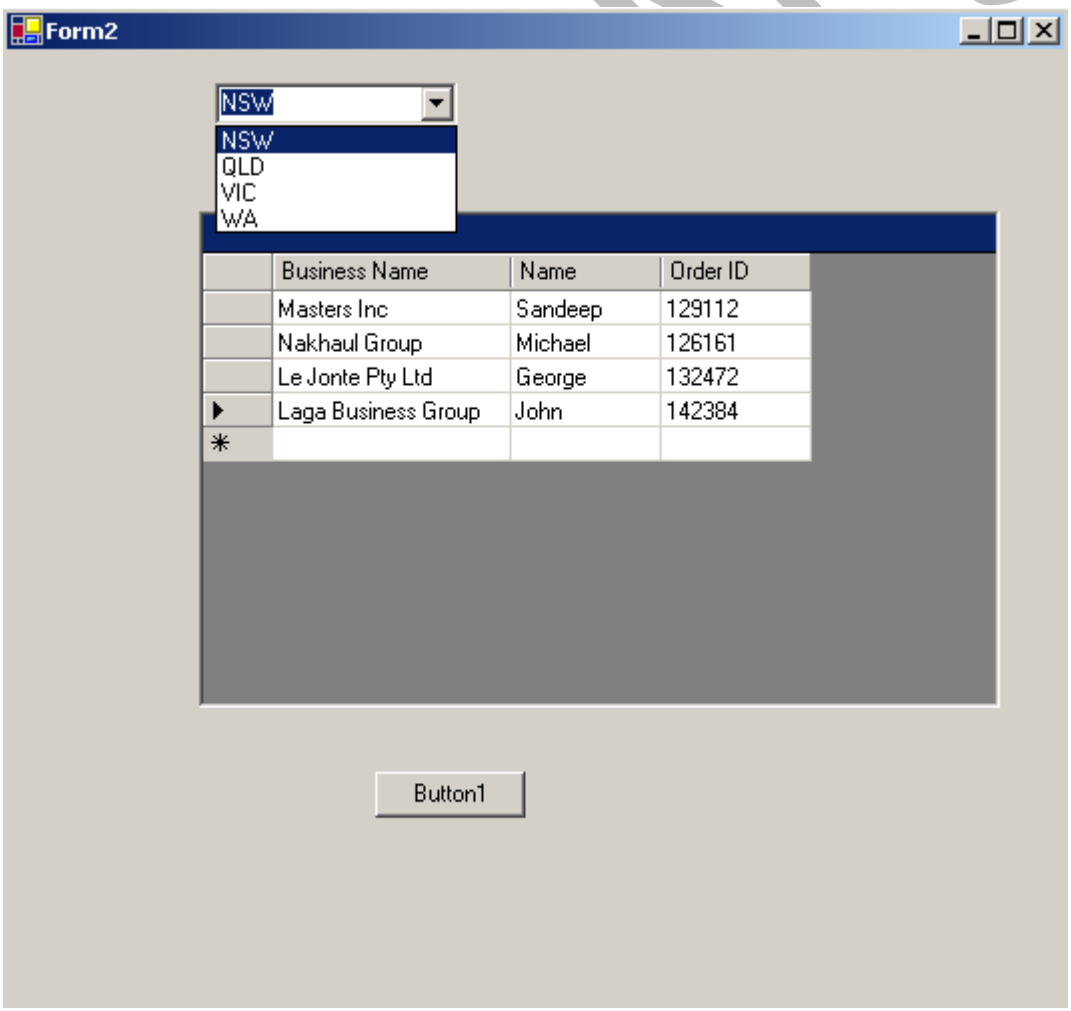
Note the above said line of code. Here, we are using the SQL Parameter indicated by the ? mark for the state field in a WHERE clause in the SQL for the second data adapter. The Sql Parameter used will display data related to the state we select in the combo box in the data grid. After you finish configuring the data adapter, you need to generate the dataset. To generate a dataset, DataSet21 select Data->Generate Dataset from the main menu.

Now, drag a ComboBox, a Button and a DataGrid control onto a new form. Select the ComboBox, open it's properties and set the DataSource property to DataSet11 andDisplayMember property to Table1.State. Select the DataGrid and in the properties window set the DataMember property to Table1 and DataSource property to DataSet21. Switch to code view and paste following code.

```
Private Sub Form2_Load(ByVal sender As System.Object,_
```

```
ByVal e As System.EventArgs) Handles MyBase.Load
DataSet11.Clear()
OleDbDataAdapter1.Fill(DataSet11)
End Sub

Private Sub Button1_Click(ByVal sender As System.Object,_
ByVal e As System.EventArgs) Handles Button1.Click
OleDbDataAdapter2.SelectCommand.Parameters("state").Value =
ComboBox1.Text
'placing a value into the SQL parameter corresponding to the state field
DataSet21.Clear()
OleDbDataAdapter2.Fill(DataSet21)
End Sub
```

After you are done with the code run the form, select a state from the ComboBox and click the button. The data relating to that particluar state will be displayed in the datagrid as shown in the image below.

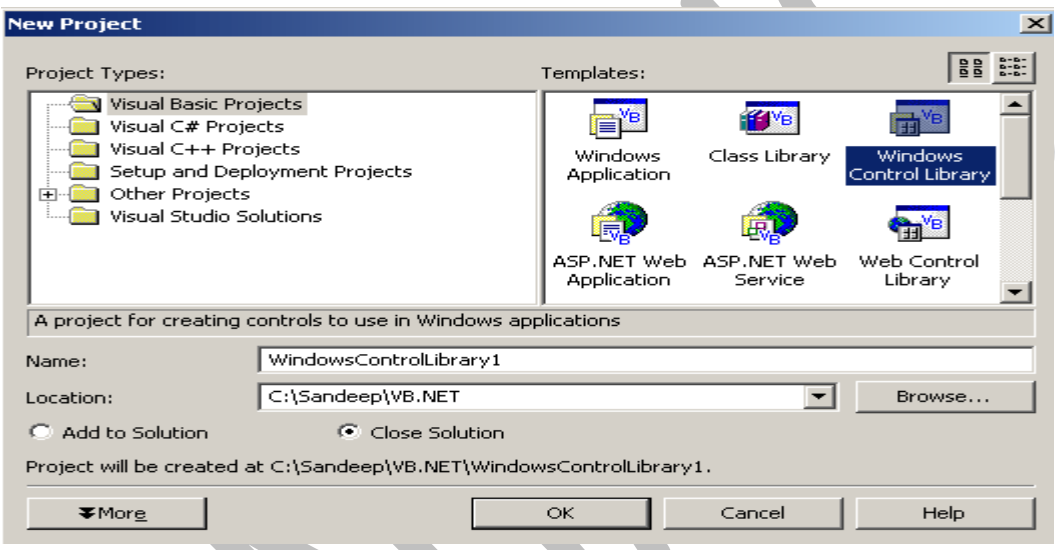


User Controls

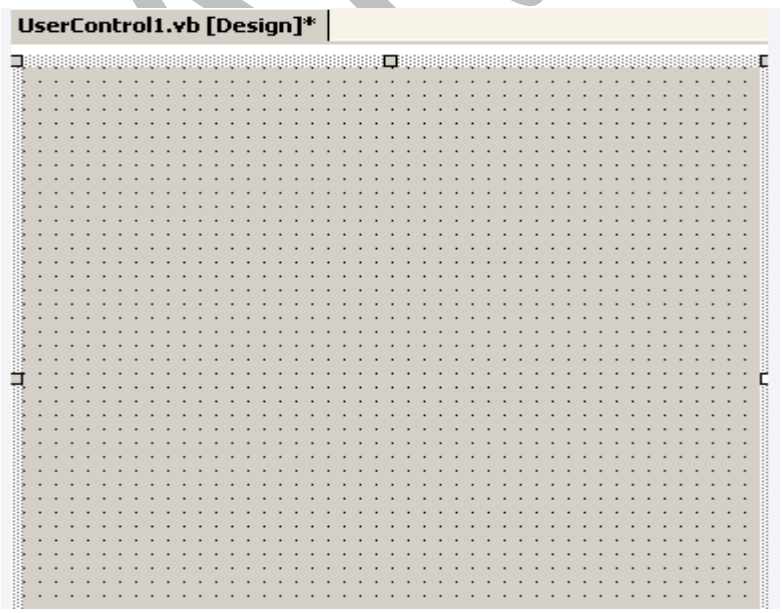
User Controls are the controls which are created by the user and they are based on the class System.Windows.Forms.UserControl. Like standard controls, user controls support properties, methods and events. Once a user control is created it can be added to any form or any number of forms like all other controls.

Creating a User Control

To create a user control select File->New->Project->Visual Basic Projects and select Windows Control Library from the templates and click OK. Alternatively, you can add user control to the existing project by selecting Project->Add User Control. The image below displays the new project dialogue to add a User Control project.

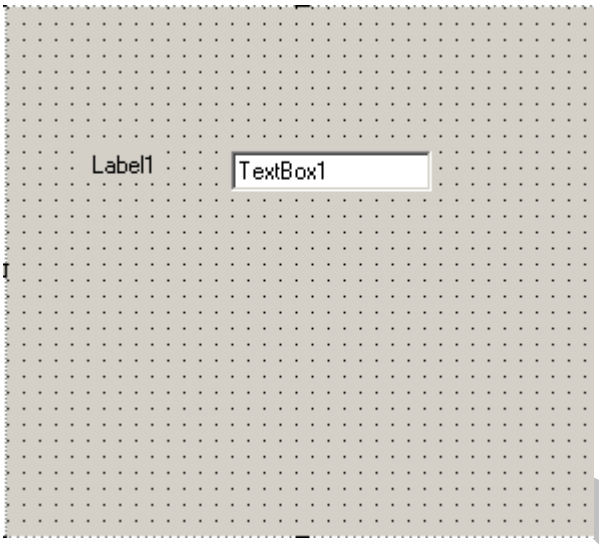


The form that opens after clicking OK looks like the image below. It looks similar to a normal form.



Creating a User Control with Example

Drag a Label and a TextBox control from the toolbox onto the new user control form. The image below displays that.

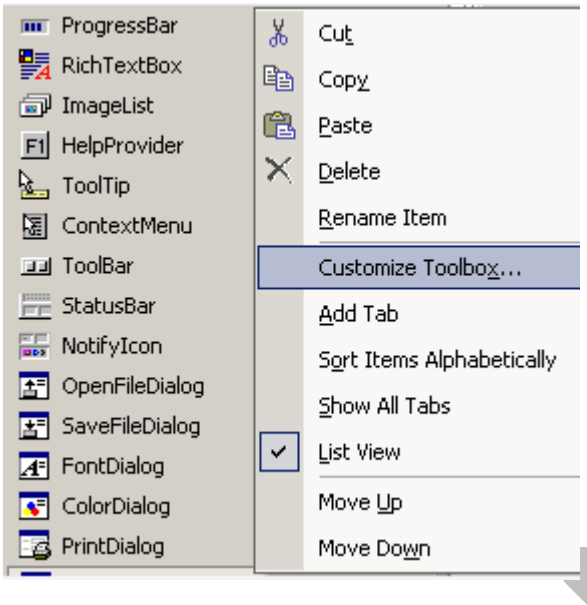


Double-click the user control form to open it's code behind file. In the code behind file type the following code below the Load event (under End Sub) of UserControl1 to set the property of the user control which is being created.

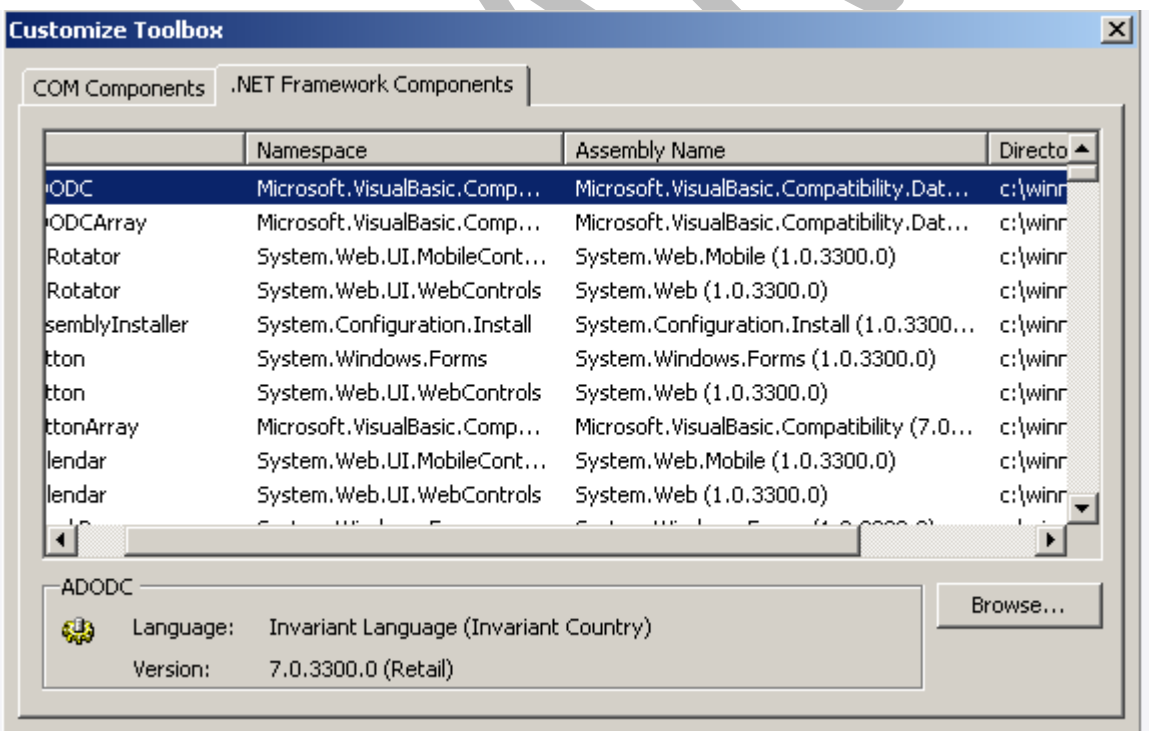
```
Public Property sanText() As String
Get
sanText = TextBox1.Text
End Get
Set(ByVal Value As String)
TextBox1.Text = Value
End Set
End Property
Public Property sanLbl() As String
Get
sanLbl = Label1.Text
End Get
Set(ByVal Value As String)
Label1.Text = Value
End Set
End Property
```

The above code implements the sanText() and sanLbl() properties with a property get/set pair. Once typing the code is done build the solution using Build->Build Solutionfrom the main menu to create the .dll (Dynamic Link Library) file to which we refer to. The dll file also makes this user control available to other projects. After finishing building the solution we next need to add this user control to the toolbox to make it available to other projects and forms. To do that, open a windows form or use an existing form. Right-click on the Windows

Form tab section on the toolbox and click on Customize Toolbox as shown in the image below.



Clicking on Customize Toolbox opens "Customize Toolbox" dialogue box as shown in the image below.



On this window select the .NET Framework components tab which displays a list. Click browse to add the dll of the user control we created and select the dll file from the bin directory of the Windows Control library project. Once that is done, click OK. The dll of the user control gets added to the .NET Framework Components. Select that and click OK. The user control which we created gets added to the toolbox. Now this control can be used like other controls in the toolbox.

Alternatively, the user control can be added to the toolbox in another way. To do that, right-click on the References item in the Solution Explorer window and select Add Reference which opens the Add Reference dialogue box. Select the projects tab and double-click the User Control item. That adds item to the selected components box at the bottom of the dialog. click OK to finish.

Using the User Control

The user control, UserControl11 which we created can be added to any form like all other controls. Open a new form and drag this user control from toolbox to the form. Drag a Button from the toolbox and place that on the form. Double-click Button to open it's Click event. Place the following code in the Click event of the Button:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal_
e As System.EventArgs) Handles Button1.Click
MessageBox.Show("TextBox" & UserControl11.sanText())
MessageBox.Show("Label1" & UserControl11.sanLbl())
End Sub
```

Run the code and enter some text in the TextBox. When you click the Button, the text entered in the TextBox is displayed in a MessageBox. That's the way User Controls are used after they are created.

Inherting Controls

One of the easiest ways to create a new control is to inherit from a preexisting control. When inheriting from an existing control, the newly created control contains all the functionality of the base control but can also serve as a base for adding new functionality. In addition to having all the functionality of the base control an inherited control also inherits the visual representation of the base control. For example, you can inherit from a TextBox control and implement code to change its appearance.

Unless sealed (NotInheritable), most Windows Controls can be used as a base class for inheritance. Inheriting from an existing control is the most easiest way of creating a new control. You can choose this method if you want to replicate an exisiting windows control and want to add more functionality to it. You can also use this method if you want to have all the functionality of the existing control but want to provide a new look and feel.

To create a new Inherited control you must specify a Windows Form control as a base class for this control. The following sample code creates a textbox that accepts only numbers as user input.To implement that functionality you must override the OnKeyPress method as shown below.

```
Public Class NumberOnlyBox Inherits System.Windows.Forms.TextBox

Protected Overrides Sub OnKeyPress(ByVal e as KeyPressEventArgs)
If Char.IsNumber(e.KeyChar)=False Then
e.handled=True
End If
```

```
End Sub

End Class
```

Files in VB .NET

Working with Files

File handling in Visual Basic is based on System.IO namespace with a class library that supports string, character and file manipulation. These classes contain properties, methods and events for creating, copying, moving, and deleting files. Since both strings and numeric data types are supported, they also allow us to incorporate data types in files. The most commonly used classes are FileStream, BinaryReader, BinaryWriter, StreamReader and StreamWriter.

FileStream Class

This class provides access to standard input and output files. We use the members of FileAccess, FileMode and FileShare Enumerations with the constructors of this class to create or open a file. After a file is opened it's FileStream object can be passed to the Binary Reader, BinaryWriter, Streamreader and StreamWriter classes to work with the data in the file. We can also use the FileStreamSeek method to move to various locations in a file which allows to break a file into records each of the same length.

StreamReader and StreamWriter Class

The StreamReader and StreamWriter classes enables us to read or write a sequential stream of characters to or from a file.

BinaryReader and BinaryWriter Class

The BinaryReader and BinaryWriter classes enable us to read and write binary data, raw 0's and 1's, the form in which data is stored on the computer.

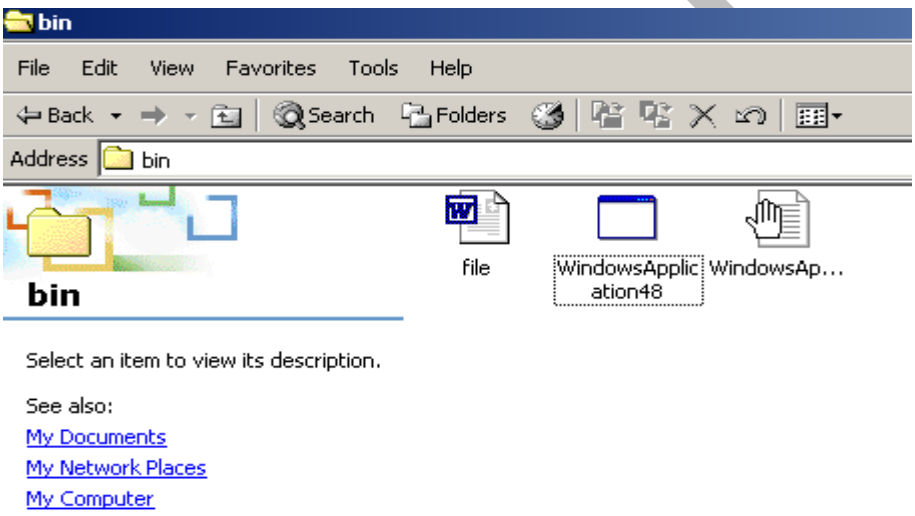
The following examples puts some code to work with textual data using FileStream and StreamReader and StreamWriter classes.

Code to create a File

```
Imports System.IO
'NameSpace required to be imported to work with files
Public Class Form1 Inherits System.Windows.Forms.Form
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles MyBase.Load
Dim fs as New FileStream("file.doc", FileMode.Create, FileAccess.Write)
'declaring a FileStream and creating a word document file named file with
'access mode of writing
Dim s as new StreamWriter(fs)
'creating a new StreamWriter and passing the filestream object fs as
```

```
argument
s.BaseStream.Seek(0,SeekOrigin.End)
'the seek method is used to move the cursor to next position to avoid text
to be
'overwritten
s.WriteLine("This is an example of using file handling concepts in VB
.NET.")
s.WriteLine("This concept is interesting.")
'writing text to the newly created file
s.Close()
'closing the file
End Sub
End Class
```

The default location where the files we create are saved is the bin directory of the Windows Application with which we are working. The image below displays that.



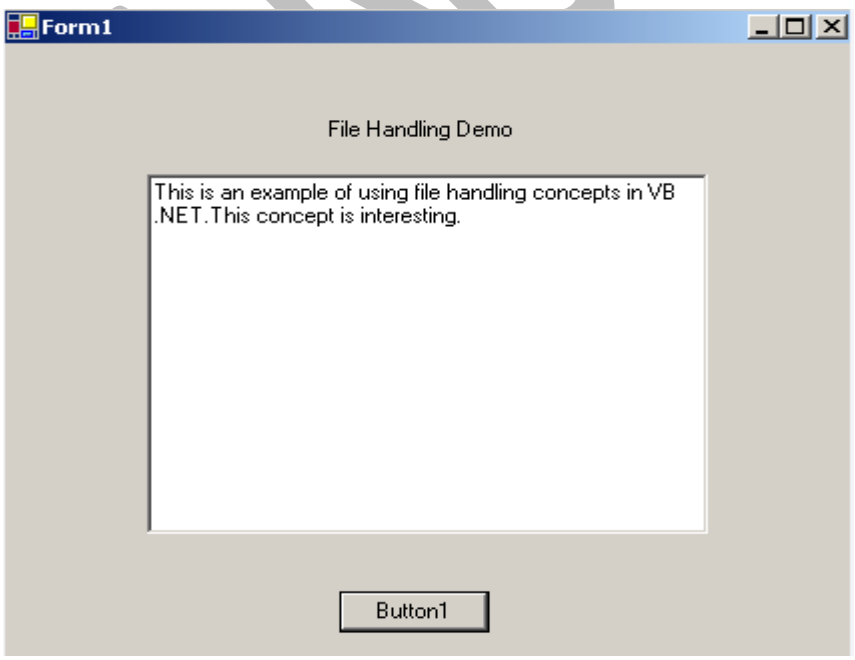
Code to create a file and read from it

Drag a Button and a RichTextBox control onto the form. Paste the following code which is shown below.

```
Imports System.IO
'NameSpace required to be imported to work with files
Public Class Form1 Inherits System.Windows.Forms.Form
Private Sub Button1_Click(ByVal....., Byval.....)Handles Button1.Click
Dim fs as New FileStream("file.doc", FileMode.Create, FileAccess.Write)
'declaring a FileStream and creating a document file named file with
'access mode of writing
Dim s as new StreamWriter(fs)
```

```
'creating a new StreamWriter and passing the filestream object fs as
argument
s.WriteLine("This is an example of using file handling concepts in VB
.NET.")
s.WriteLine("This concept is interesting.")
'writing text to the newly created file
s.Close()
'closing the file

fs=New FileStream("file.doc",FileMode.Open,FileAccess.Read)
'declaring a FileStream to open the file named file.doc with access mode
of reading
Dim d as new StreamReader(fs)
'creating a new StreamReader and passing the filestream object fs as
argument
d.BaseStream.Seek(0,SeekOrigin.Begin)
'Seek method is used to move the cursor to different positions in a file, in
this code, to
'the beginning
while d.peek()>-1
'peek method of StreamReader object tells how much more data is left in
the file
RichTextbox1.Text &= d.readLine()
'displaying text from doc file in the RichTextBox
End while
d.close()
End Sub
```

The image below displays output of the above code.



Working with Directories

We will work with the File and Directory classes in this section. We will create a directory and copy a file into the newly created directory.

File Class

The File class in VB .NET allows us to work with files, allowing to copy, delete and create files.

Directory Class

The Directory class in VB .NET allows us to create and work with Folders/Directories. With this class we can create, edit and delete folders and also maintain drives on the machine.

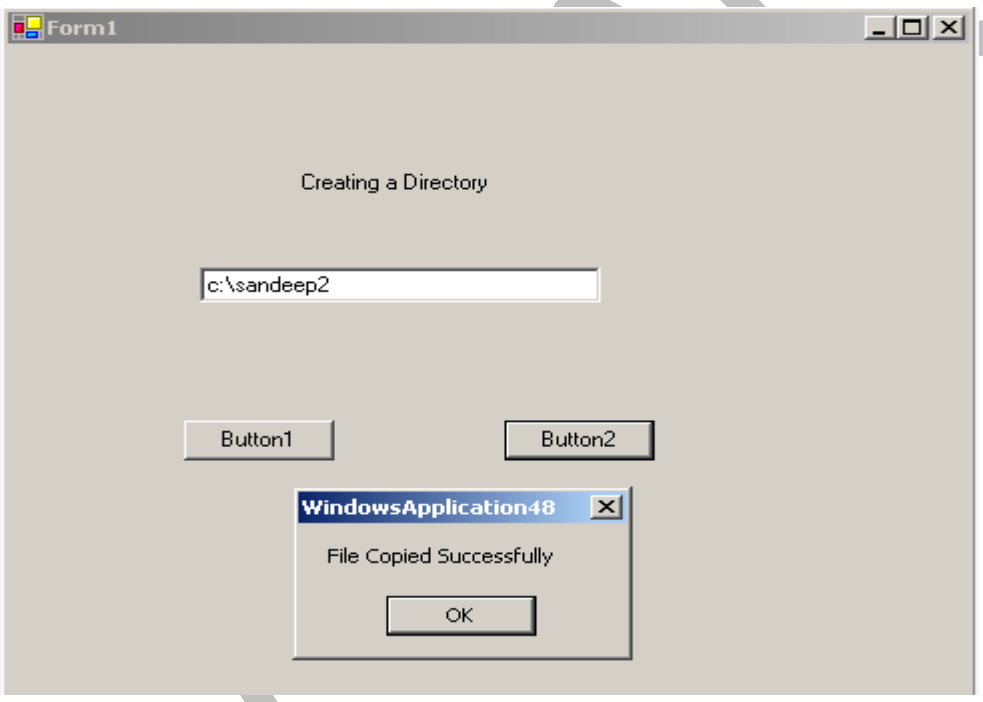
Code to work with File and Directory Class

The following code will create a directory and copy a file into that new directory. To create a new directory we should use the Directory class's CreateDirectory method. Drag two Button's (Button1, Button2), a TextBox and a OpenFileDialog control from the toolbar onto the Form. We will use the TextBox to specify a location to create the Directory when Button1 is clicked and the OpenFileDialog control to open a file and copy it into the newly created Directory when Button2 is clicked. The namespace to be imported isSystem.IO. The code for that looks like this:

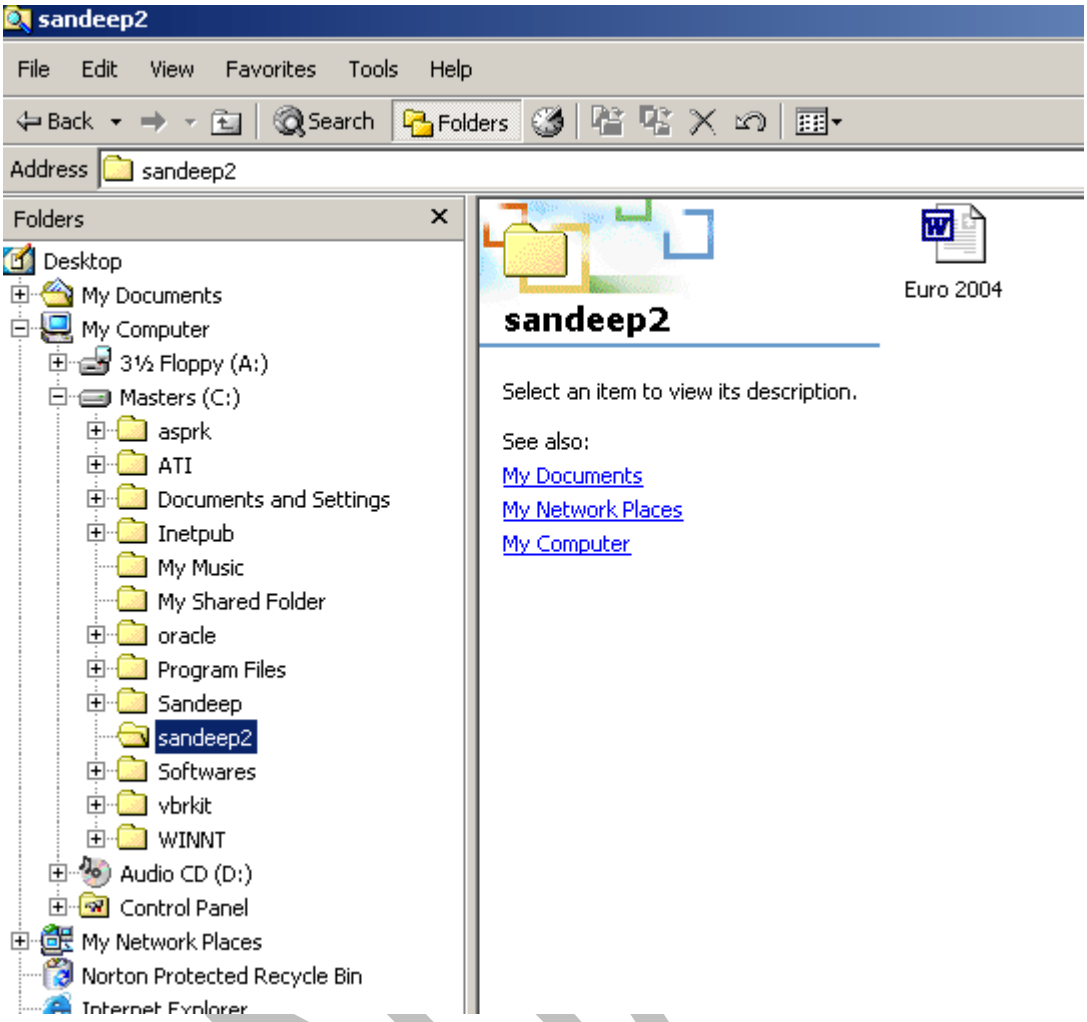
```
Imports System.IO
Public Class Form1 Inherits System.Windows.Forms.Form

 'Windows Form Designer Generated Code

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles MyBase.Load

End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click

Try
Directory.CreateDirectory(TextBox1.Text)
 'Creating a directory by specifying a path in the TextBox, of the form c:\examples
'Instead of using a TextBox you can directly type the location of the directory like
this
'Directory.CreateDirectory("c:\examples")
Catch
End Try
MsgBox("Done")
End Sub
```

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button2.Click
Try
If OpenFileDialog1.ShowDialog <> DialogResult.Cancel Then
File.Copy(OpenFileDialog1.FileName, TextBox1.Text & "\" & _
OpenFileDialog1.FileName.Substring(OpenFileDialog1.FileName.LastIndexOf("\")))
'The above line of code uses OpenFileDialog control to open a dialog box where you
'can select a file to copy into the newly created directory
End If
Catch
End Try
MsgBox("File Copied Successfully")
End Sub
End Class
```

That's all it takes to create a directory and copy a file into the newly created directory in VB.NET. The image below shows output from above code.



The image below shows the directory I created and the file I copied to the new directory using the code above.



Multithreading

Multithreading gives programs the ability to do several things at a time. Each stream of execution is called a thread. Multithreading is used to divide lengthy tasks into different segments that would otherwise abort programs. Threads are mainly used to utilize the processor to a maximum extent by avoiding it's idle time. Threading lets a program seem as if it is executing several tasks at once. What actually happens is, the time gets divided by the computer into parts and when a new thread starts, that thread gets a portion of the divided time. Threads in VB .NET are based on the namespace System.Threading.

Creating Threads

To create threads lets work with an example. The following example is an extract from Steven Holzner's reference, Programming with Visual Basic.NET- Black Book. Open a new windows application and name it as Thread and add a class named count1 using the Projects->Add Class item. This class will count from 1 to a specified value in a data member named CountTo when you call the Count method. After the count has reached the value in CountTo, a FinishedCounting event will occur. The code for the Count class looks like this:

```
Public Class Count1
```

```
Public CountTo as Integer
Public event FinishedCounting(By Val NumberOfMatches as Integer)
Sub Count()
Dim ind,tot as Integer
tot=0
For ind=1 to CountTo
tot+=1
Next ind
RaiseEvent FinishedCounting(tot)
'makes the FinishedCounting event to occur
End Sub
End Class
```

Let's use this class with a new thread. Get back to the main form and create an object of this class, counter1, and a new thread, Thread1. The code looks like this:

```
Public Class Form1 Inherits System.Windows.Forms.Form
Dim counter1 as new Count1()
Dim Thread1 as New System.Threading.Thread(Address of
counter.Count)
```

Drag a Button and two TextBoxes (TextBox1, TextBox2) onto the form. Enter a number in TextBox1. The reason for entering a number in textbox is to allow the code to read the value specified in TextBox1 and display that value in TextBox2, with threading. The code for that looks like this:

```
Public Class Form1 Inherits System.Windows.Forms.Form
Dim counter1 as new Count1()
Dim Thread1 as New System.Threading.Thread(Address of
counter.Count)
Private Sub Button1_Click(ByVal sender as System.Object, ByVal e as
System.EventArgs)_
Handles Button1.Click
TextBox2.Text=" "
counter1.CountTo=TextBox1.Text
AddHandler
counter1.FinishedCounting,AddressOfFinishedCountingEventHandler
'adding handler to handle FinishedCounting Event
Thread1.Start()
'starting the thread
End Sub
Sub FinishedCountingEventHandler(ByVal Count as Integer)
'FinishedCountingEventHandler
TextBox2.Text=Count
End Sub
```

The result of the above code displays the value entered in TextBox1, in TextBox2 with the difference being the Thread counting the value from 1 to the value entered in TextBox1.

Multithreading

Suspending a Thread

Threads can be suspended. Suspending a thread stops it temporarily. Working with the example in the previous section, add a new button Button2 to the main form. When this button is clicked the thread is suspended. The code for that looks like this:

```
Private Sub Button2_Click(ByVal sender as System.Object, ByVal e as
System.EventArgs)_
Handles Button2.Click
Thread1.Suspend()
End Sub
```

Resuming a Thread

Threads can be resumed after they are suspended. With the example above, add a new button Button3 to the main form. When this button is clicked the thread is resumedfrom suspension. The code for that looks like this:

```
Private Sub Button3_Click(ByVal sender as System.Object, ByVal e as
System.EventArgs)_
Handles Button3.Click
Thread1.Resume()
End Sub
```

Making a Thread Sleep

Threads can be made to sleep which means that they can be suspended over a specific period of time. Sleeping a thread is achieved by passing the time (inmilliseconds,1/1000 of a second) to the thread's sleep method. With the example above, add a new button Button4 to the main form. When this button is clicked the thread is stopped. The code for that looks like this:

```
Private Sub Button4_Click(ByVal sender as System.Object, ByVal e as
System.EventArgs)_
Handles Button4.Click
Thread1.Sleep(100/1000)
End Sub
```

Stopping a Thread

Threads can be stopped with it's abort method. With the example above, add a new button Button5 to the main form. When this button is clicked the thread is stopped. The code for that looks like this:

```
Private Sub Button5_Click(ByVal sender as System.Object, ByVal e as
```

```
System.EventArgs)_
Handles Button5.Click
Thread1.Abort()
End Sub
```

Thread Priorities

Threads can also be assigned priority for execution. Thread priority can be set by the thread's Priority property and assigning a value from predefined Thread Priorityenumeration.

Values for Thread Priority:

- Above Normal -> Gives thread higher priority
- Below Normal ->Gives thread lower priority
- Normal -> Gives thread normal priority
- Lowest -> Gives thread lowest priority
- Highest -> Gives thread highest priority

Working with the above example, add a new button Button6 to the main form. When this button is clicked the thread is assigned Highest priority .The code for that lookslike this:

```
Private Sub Button6_Click(ByVal sender as System.Object, ByVal e as
System.EventArgs)_
Handles Button6.Click
Thread1.Priority=System.Threading.ThreadPriority.Highest
'setting Highest priority for the thread
End Sub
```

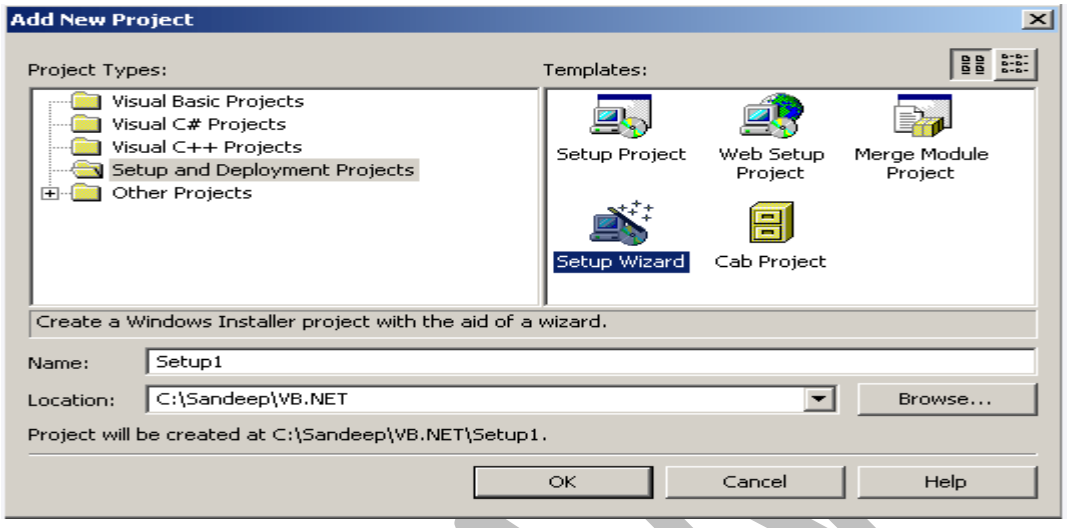
Deploying Applications

Once an application is developed and if we want to distribute that application, we need to deploy that. Deployment is the process where we create an executable file which can be installed on any machine where the application can run. We can use the built-in deployment feature that comes with Visual Basic to create a Windows Installer file - a.msi file for the purpose of deploying applications.

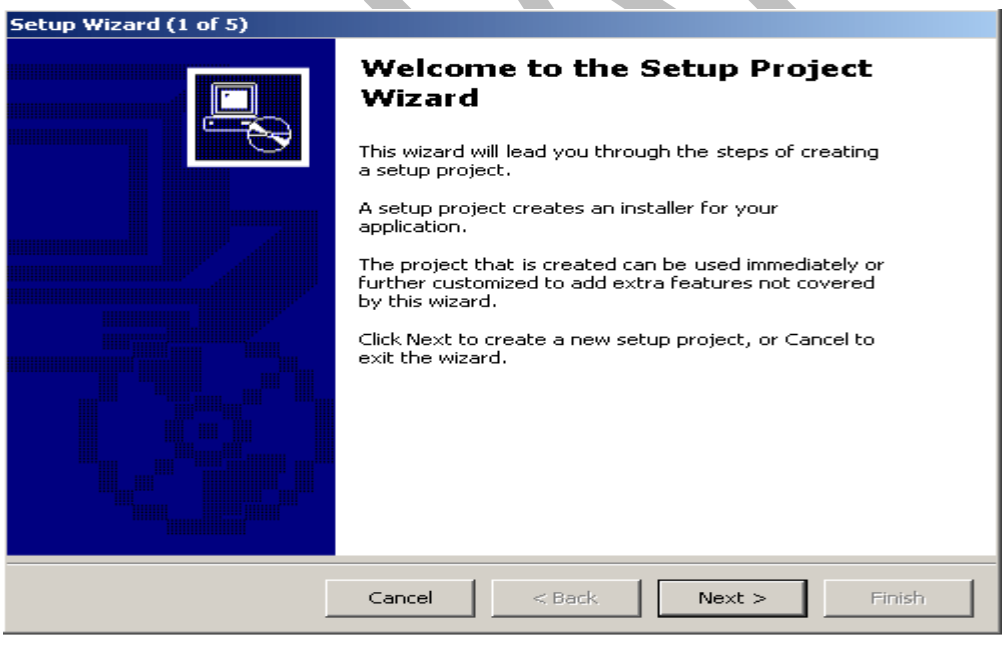
Let's look at the process with an example. Let's assume we have a form with a TextBox and a Button. When the Button is clicked the TextBox should display "Thisapplication is Deployed". Let's name this application as Deploy. The code for the click event of the Button looks like this:

```
Private Sub Button1_Click(By Val sender as System.Object, By Val e_
as System.EventArgs)Handles Button1.Click
TextBox1.Text="This application is Deployed"
End Sub
```

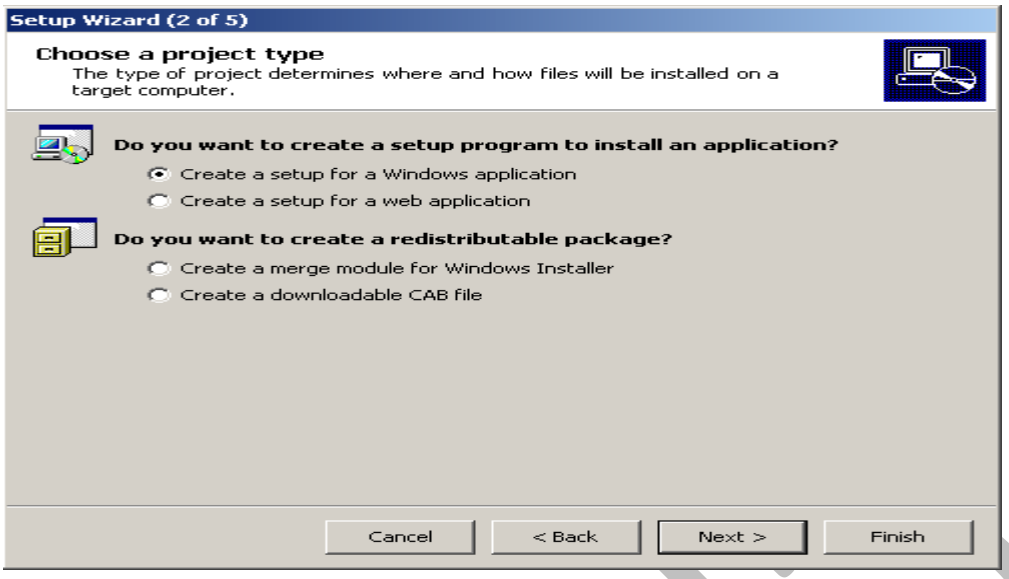
Next, we need to create an executable (exe) file for this application. To do that select Build->Build from the main menu which builds Deploy.exe. Next, we need to create aninstaller file for Deploy (which is the example) which is a file with .msi extension. To do that, select File->Add Project->New Project which opens the new project dialogue. Select "Setup and Deployment Projects" icon in the projects type box and Setup Wizard in the templates box. It looks like the image below.



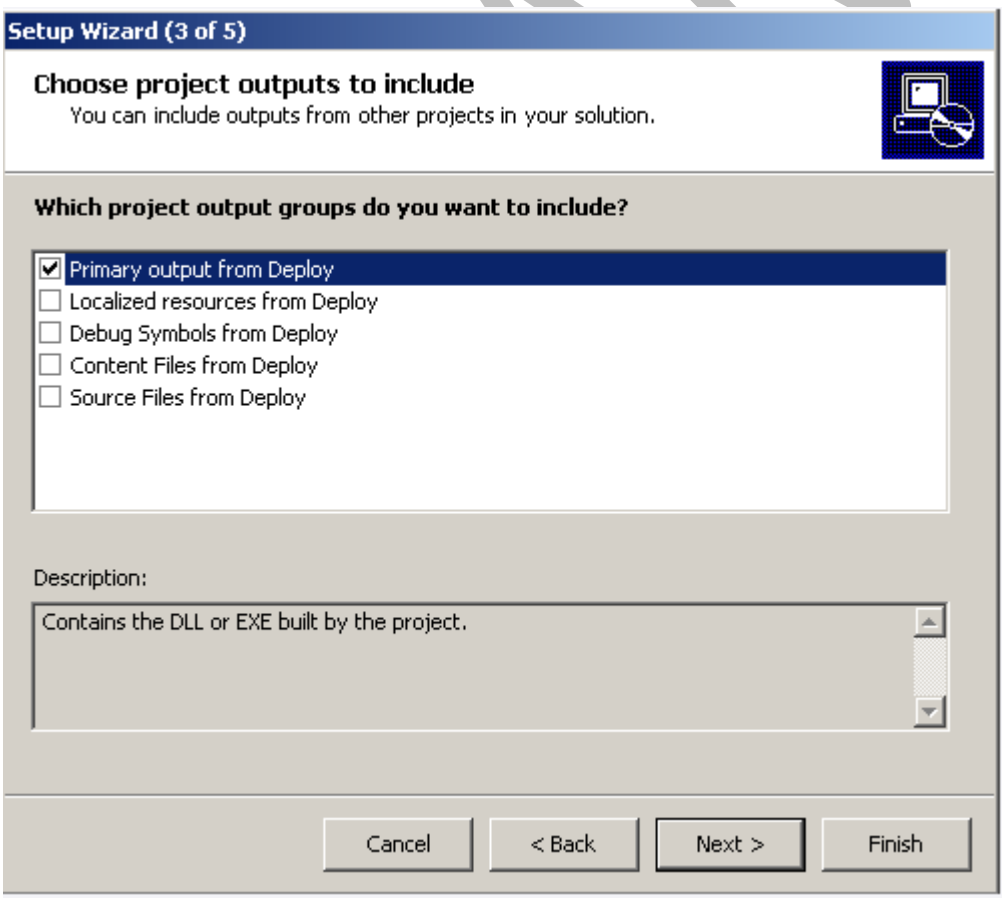
Click OK to open the Setup Wizard. The Setup wizard window looks like the image below.



Click next on the above pane to take you to second pane in the Wizard. The new pane allows us to create deployment projects both for Windows and Web Applications. Here, select the radio button which says "Create a setup for Windows Application" as this is deploying a windows application and click next. It looks like the image below.

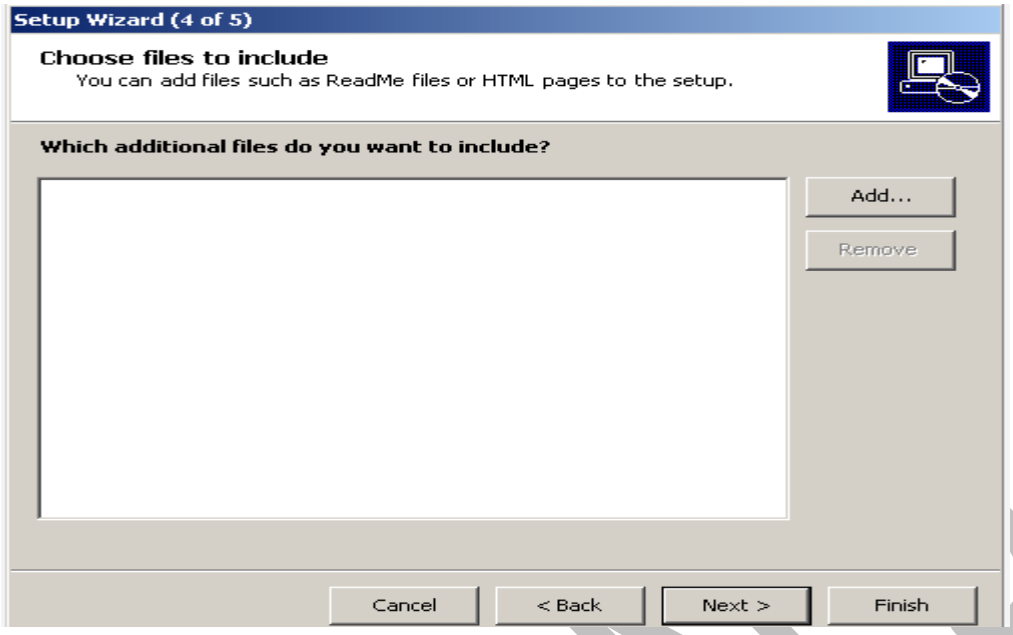


Clicking next opens a new pane which has options like Deploying only primary output from the project or both the project and source code or content files. Check the checkbox which you want, in this case check the checkbox that says "Primary Output from Deploy" and click next. It looks like the image below.

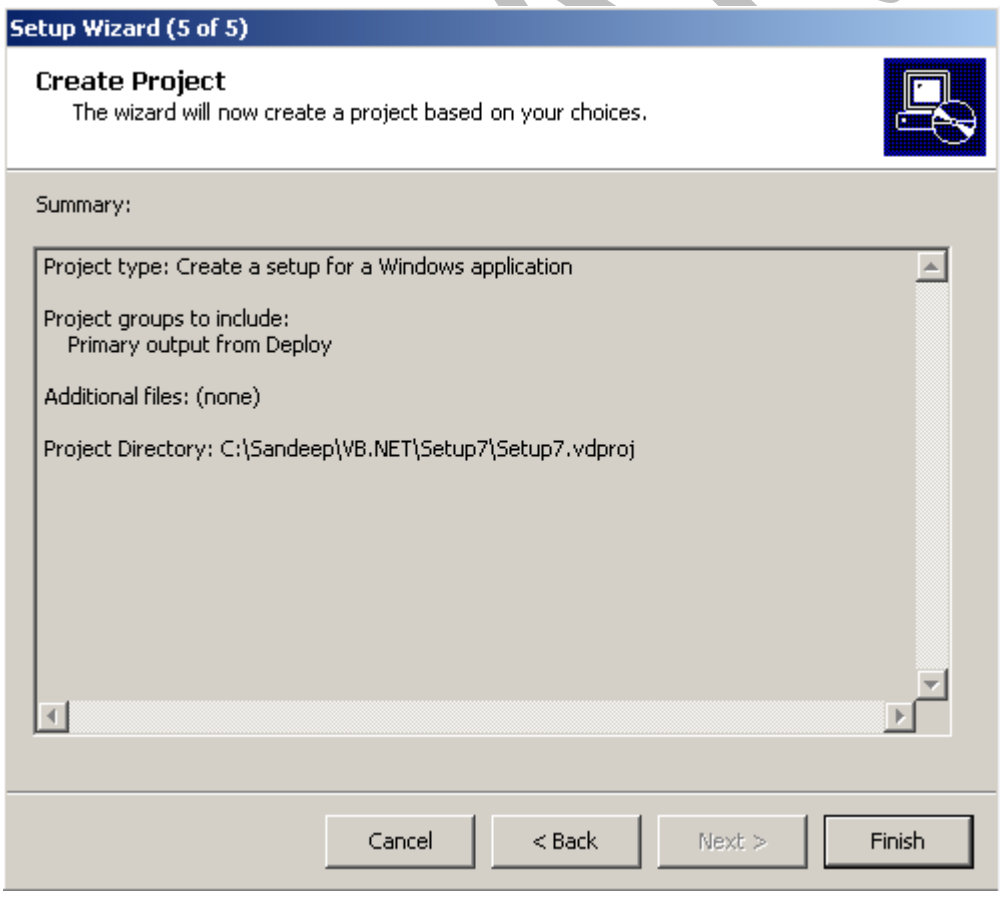


Clicking next opens a new pane which asks if you want any additional files to be added. If you wish, you can include other files, like an icon for the application. In this example don't

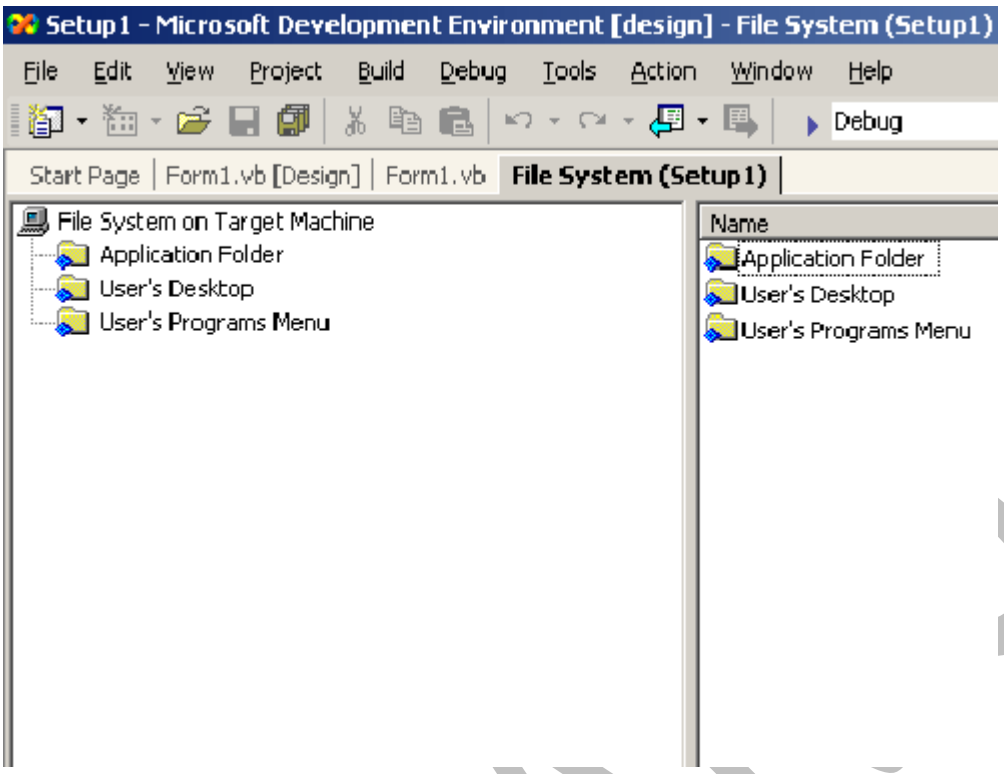
include any files and click next. It looks like the image below.



Doing that brings up the last pane of the Setup Wizard which looks like the image below. Click Finish on this pane.



Clicking finish opens up a File System window which looks like the image below.

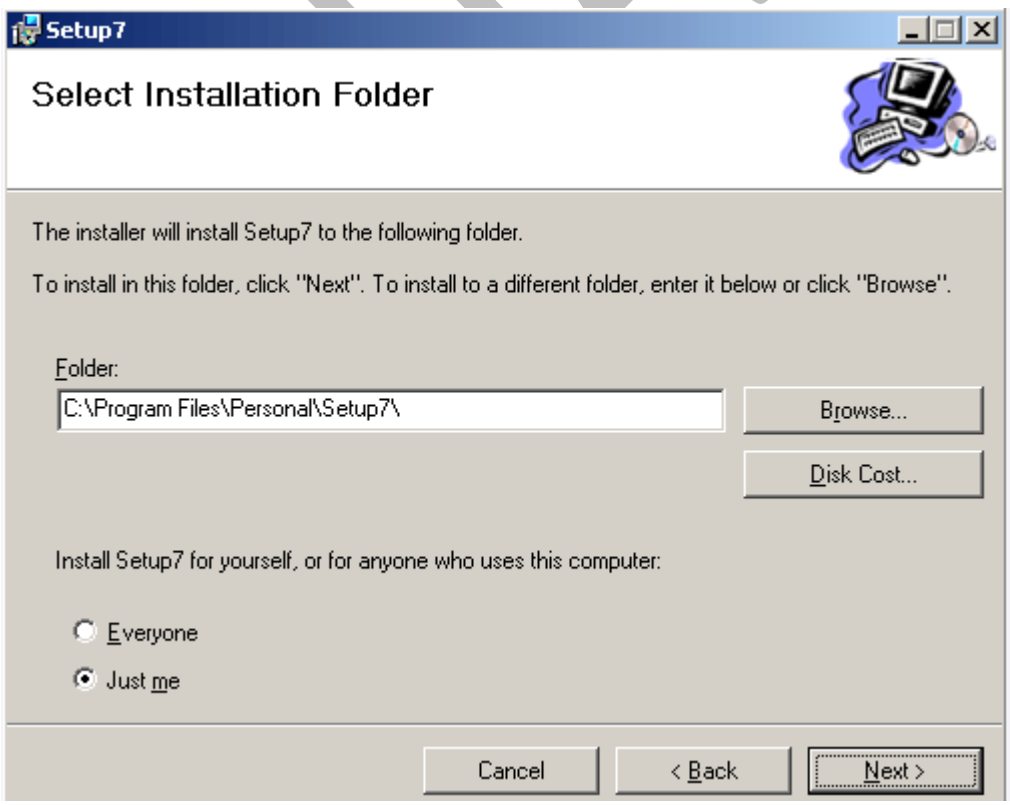


This window allows us to create shortcuts to the application on the desktop and in our Programs Menu. To create a shortcut, right-click on the folder "User's Desktop" and select "Create Shortcut to User's Desktop". Rename the shortcut to "Deployment". If we want a shortcut to the application from our Programs Menu, right-click on "User's Program Menu" folder and select "Create Shortcut to User's Program Menu". Rename the shortcut to "Deployment". Once you are finished with it, click on "Application Folder" and open it's properties. In the properties window set the property "Always Create" to True. Set the same for "User's Desktop" and "User's Programs Menu" folders. If you want any additional information to include with the set-up project, like the manufacturer, author etc, click on Setup1 project and open it's properties. You can set additional information here. Once your are done with it build the project by right-clicking on Setup1 and selecting Build. This builds the application. The setup file is created in the debug folder of Setup1 project.

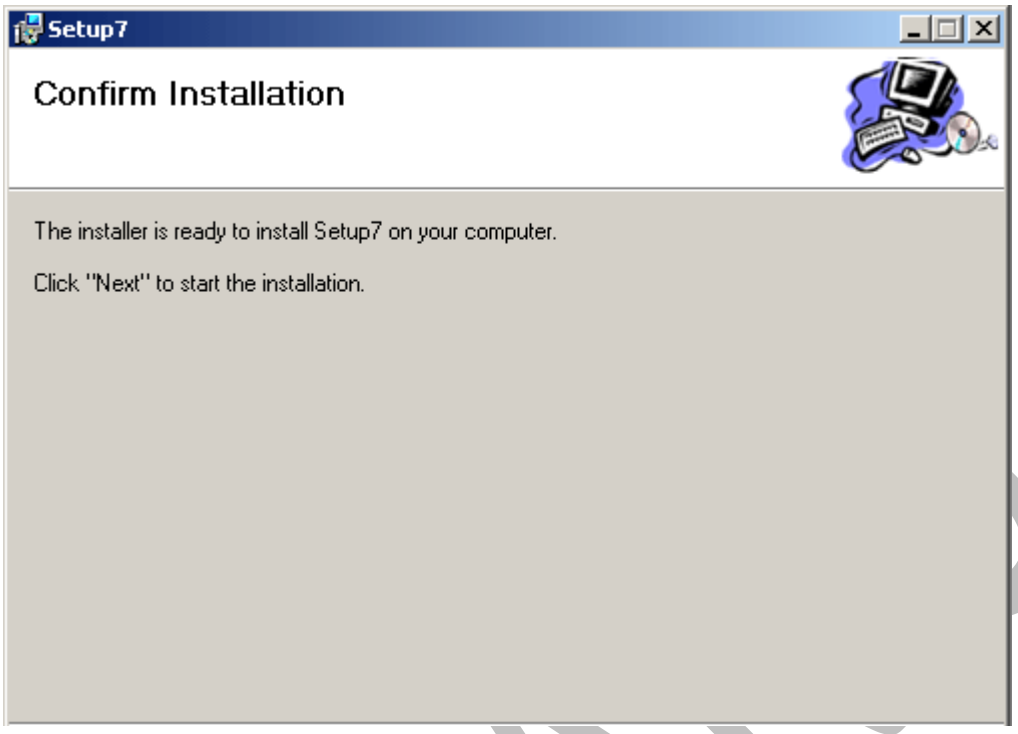
Deploying the Application

To deploy the application we need to copy Setup1.msi file to the target machine. Once copying is done,
double-click that file which opens the Windows Installer which is a new window which says "Welcome to Setup1 Setup Wizard". It looks like the image below (mine was Setup7, yours will be Setup1).

Click next to move to next window which allows us to specify the location where the application should be installed. It looks like the image below.



Select the location for installation and click next. Clicking next installs the application. The confirmation window looks like the image below.



Now, double-click the newly installed Deployment.exe file to run and get the desired result. You can select that from your Programs Menu or Desktop. That completes the process of Deploying Applications.

Make sure the Target Machine on which the application will be installed supports Windows Installer and .NET Framework.

XCOPY Deployment

In addition to the deployment tools included in Visual Studio .NET there are other alternative methods for deploying applications. In most cases, the deployment tools provide more robust installation. For some simple cases the alternative methods may be adequate.

XCOPY-deployment enables applications to be deployed to client machines simply by copying files into the desired application directory. With this method no complicated setup scripts or interactions with the system registry are required. In addition, the auto-downloading of applications for Windows makes the deployment of rich Windows-based applications as easy as deploying a Web page.

Xcopy Command

The DOS Xcopy command is a simple way to copy a project or application from one location to another. But it is recommended that you deploy your project rather than using Xcopy. The Xcopy command does not register or verify the location of assemblies. More importantly, using Xcopy to deploy an application will not take advantage of WindowsInstaller features, making it possible to overwrite files that could cause other applications to break.

To see the command-line syntax and options for the Xcopy command, type Xcopy /? in the Visual Studio command-prompt window.

XML Web Services

A Web Service (XML Web Service) is a unit of code that can be activated using HTTP requests. Stated another way, a Web Service is an application component that can beremotely callable using standard Internet Protocols such as HTTP and XML. One more definition can be, a Web Service is a programmable URL. Web Services came into existence to deliver distributed computing over the Internet. A major advantage of the Web services architecture is, it allows programs written in different languages on different platforms to communicate with each other in a standards-based way. Simply said, a Web service is a software service exposed on the Web through SOAP, described with a WSDL file and registered in UDDI.

Why XML Web Services?

Today, available technologies like Component Object Model (COM), Remote Method Invocation (RMI), Common Object Request Broker Architecture (CORBA) and Internet Inter-ORB Protocol (IIOP) are used to package application logic into reusable components and are called remotely across multiple platforms. The Internet today consists of tremendous number of heterogeneous systems and a key limitation of all the above said technologies is they are not easily interoperable with these different systems and that limits their effectiveness as a standard method for programming the Web. This is because of the dependencies of these technologies on a particular language or a particular Operating System or Object-model specific protocols. Web Services on the other hand are very different when compared to the said technologies as they are built upon widely accepted standards that can interoperate easily on the Internet. A key to the success of Web Services is that they use a text-based messaging model to communicate which allows them to operate effectively on different platforms.

Example of a Web Service

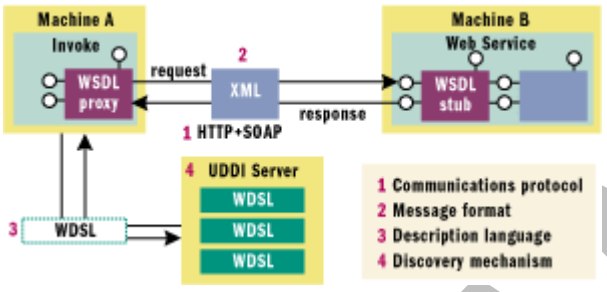
There is already an example in the "**.NET Defined**" section of this Web site. Here is another example similar to that. Consider a commerce site that allows consumers to shop online. After all the shopping has been done this site calculates all the charges and the shipping costs based on a variety of shipping options. This site will have alliances with different shipping companies to ship the products to it's consumers. This site might maintain a set of database tables that describe the shipping options and costs for each shipping company based on their location and services. With this approach, whenever there is a change in shipping options or costs of an existing shipping company change or if a new shipping company forms an alliance with this commerce site and provides it's services the Webmaster of the commerce site has to restructure the database and update them to fit the changes. This approach is not only time consuming but also requires the commerce site to invest extra IT costs to maintain it's database. Now, imagine this commerce site programmatically calling a Web Service on it's site provided by the shipping company. What happens with this approach is, the commerce site can calculate shipping costs based on the shipping option that a consumer specifies in his request and returns the costs in real time. This approach eliminates the need for the commerce site to maintain a separate database table for shipping companies and also

all the shipping costs are calculated on the shipping company site and returned to the commerce site in real time.

Some other applications of Web Services are:

- Information sources like stock quotes, weather forecasts, sports scores etc that could easily incorporate into applications
- Services that provide commonly needed functionality for other services. Example, user authentication, usage billing etc
- Services that integrate a business system with other partners

The image below shows Web Services Architecture.



Foundational elements of Web Services

The .NET Framework provides an excellent foundation for building and consuming Web Services. A key to the broad-reach capabilities of these Web Services is a foundation built on Internet Standards that does not rely on any platform, protocol or OS. This foundation provides the following capabilities to Web Services:

- A standard method for describing data
- A standard message format for communicating request and response
- A standard method for describing the capabilities of Web Services
- A method to discover what Web Services are available at any site
- A method to describe what sites provide Web Services
- **Web Services Infrastructure**
- In this section we will know the infrastructure that is needed to support Web Services. The four primary infrastructure pieces needed are: Web service Directories, Web service Discovery, Web service Description and Web service Wire Formats.
- **Web Service Directories**
- Web service Directories allow us to locate providers of Web services. They provide a centralized,
  Internet-accessible location where Web service users (consumers) can easily locate services offered by other companies and organizations. They can be called as "Yellow Pages" of Web services where we can find a list of Web services and their locations. Using these directories we can search and find any Web service based on the type ofservice we need.
- The Universal Description, Discovery and Integration (UDDI) currently is the de facto standard for cataloging and finding Web services. The UDDI organization created a directory of services, API's etc for participating companies and organizations providing Web services. You can visit the UDDI website to search for

Web services. Else, you can use Visual Studio .NET 's web reference feature to search these directories.

- **Web Service Discovery**
- Web services Discovery provides the capability to locate Web services. It's a process of locating documents that define a specific service. These capabilities are described in a standard way using the Web Services Description Language (WSDL) which is specifically designed for this. The discovery process allows a Web service user to search and locate the WSDL document. The DISCO (discovery) specification defines the existence of Web services and helps to locate the Web service's WSDL document. DISCO documents are XML based and have a file extension of .vsdisco. The discovery document is a container for two elements, pointers to WSDL document and pointers to other discovery documents. These pointers are in the form a URL.
- You can use Visual Studio .NET 's web reference feature which locates the Web services automatically using the discovery process. To do that you need to enter the URL of the discovery document which will initialize the discovery process. Else, use the .NET Framework's disco tool to search for Web service description files.
- **Web Service Description**
- Web service Description is an XML document that enables Web service capabilities to be described. Using WSDL we can clearly define the Web-addressable entry points in terms of request/response messages. Also this description includes information about the supported protocols and data types processed by the Web service. ASP .NET and the .NET platform provides the support for generation of this WSDL documents from the Web Service assembly when requested.
- The standard method of interacting with a Web Service is through the use of a proxy class. Visual Studio .NET and ASP .NET provide tools to generate a Web Service proxy class. The proxy class is similar to the actual Web service but does not contain all the implementation part. With Visual Studio .NET we can generate the proxy class from WSDL documents with it's Web reference feature to locate a Web service which we want to call. After locating the WSDL document we can generate the proxy class using the Add reference button.
- **Web Service Wire Formats**
- Web service Wire Format allow Web services to exchange data and messages. Wire formats describe the method by which Web service request/response messages are encoded and transported between the Web Service and any consumer. The three wire formats supported are : HTTP-GET, HTTP-POST and HTTP-SOAP.
- **HTTP-GET**
- The HTTP-GET protocol encodes Web service operation requests and arguments in the URL of the Web service. This is coded as part of the URL string and any arguments are coded as query string parameters appended to the base URL. The URL specifies the Web addressable entry point for the Web service which is a .asmx file.
- **HTTP-POST**
- The HTTP-POST protocol encodes Web Service operation requests and arguments within the payload area of the HTTP-POST request as name/value pairs. HTTP-POST is similar to HTTP-GET but the difference is HTTP-POST passes parameters within the actual HTTP request header rather than as a query string appended to the URL.
- **HTTP-SOAP**
- HTTP-SOAP is the default wire format. Based on the SOAP specification it supports the widest range of simple and complex data types. Web service request and response messages are encoded into SOAP message that are included in the payload area of an

HTTP-POST message, SOAP messages are encoded in XML using the SOAP vocabulary defined in the specification.

XML, SOAP, UDDI

- **XML**
- XML provides a standards-based method for describing data. XML is used extensively in building and consuming Web services. XML has the ability to describe data that is highly interoperable among many different systems on the Internet. Using the basic elements of XML we can define simple and complex data types and relationships. XML promotes the ability of Web services to communicate their data efficiently and effectively. It's this XML that ensures a consistent and accurate interpretation of the data when the service and consumer reside on different platforms.
- You can have a overview of XML in the "**Essential XML**" section of this site.
- **SOAP**
- We hear a lot about SOAP these days. Let's take a look what SOAP is and why it is related to .NET. Simple Object Access Protocol (SOAP) is a lightweight protocol for exchange of information in a decentralized, distributed environment. It's an industry-standard message format that enables message-based communications for Web services. It's XML based and consists of three parts, an envelop that defines a framework for describing what is in a message and how to process it, a set of encoding rulesfor expressing instances of application-defined data types and a convention for representing remote procedure calls (RPC). The capability of SOAP to provide a modular packaging model and encoding mechanisms for encoding data within modules allows it to be used over multiple protocols with a variety of different programming models.
- There are optional parts of the SOAP specification and one optional part defines what an HTTP message that contains a SOAP message looks like. This HTTP binding is important as HTTP is supported by almost all operating systems. The HTTP binding is optional, but almost all SOAP implementations support it as it's the only standardized protocol for SOAP. For this reason, there's a common misconception that SOAP requires HTTP. Some implementations support MSMQ, MQ Series, SMTP, or TCP/IP transports, but almost all current XML Web services use HTTP because it is ubiquitous.
- A major source of confusion when getting started with SOAP is the difference between the SOAP specification and the many implementations of the SOAP specification. Most people who use SOAP don't write SOAP messages directly but use a SOAP toolkit to create and parse the SOAP messages. These toolkits generally translate function calls from some kind of language to a SOAP message. For example, the Microsoft SOAP Toolkit 2.0 translates COM function calls to SOAP and the Apache Toolkit translates JAVA function calls to SOAP. The types of function calls and the data types of the parameters supported vary with each SOAP implementation, so a function that works with one toolkit may not work with another. This isn't a limitation of SOAP but rather of the particular implementation you are using.
- By far the most compelling feature of SOAP is that it has been implemented on many different platforms. This means that SOAP can be used to link disparate systems within and without an organization. Many attempts have been made in the past to come up with a common communications protocol that could be used for systems integration but none of them have had the widespread adoption that SOAP has. That's because SOAP is much smaller and simpler to implement than many of the previous

protocols. For example, DCE and CORBA took years to implement. SOAP, however, can use existing XML Parsers and HTTP libraries to do most of the hard work, so a SOAP implementation can be completed in a matter of months. This is why there are more than 70 SOAP implementations available. SOAP obviously doesn't do everything that DCE or CORBA do, but the lack of complexity in exchange for features is what makes SOAP so readily available.

- **UDDI**
- Universal Discovery Description and Integration is like the "Yellow Pages" of Web services. As with traditional yellow pages, we can search for a company that offers the services we need, read about the service offered and contact the company for more information. We can also offer a Web service without registering it in UDDI.
- A UDDI directory entry is an XML file that describes a business and the services it offers. There are three parts to an entry in the UDDI directory. The "white pages" describe the company offering the service, like, name, address, contacts, etc. The "yellow pages" include industrial categories based on standard taxonomies the Standard Industrial Classification. The "green pages" describe the interface to the service in enough detail for someone to write an application to use the Web service. The way services are defined is through a UDDI document called a Type Model or tModel. In many cases, the tModel contains a WSDL file that describes a SOAP interface to an XML Web service, but the tModel is flexible enough to describe almost any kind of service.
- The UDDI directory also includes several ways to search for the services we need to build our applications. For example, we can search for providers of a service in a specified geographic location or for business of a specified type. The UDDI directory will then supply information, contacts, links, and technical data to allow us to evaluate which services meet our requirements.
- UDDI allows us to find businesses we might want to obtain Web services from. If we already know whom we want to do business with but if we don't know what services are offered then we can use the the WS-Inspection specification that allows us to browse through a collection of XML Web services offered on a specific server to find which ones might meet your needs.

XML Web Services

Sample Service 1

In this section we will create a simple Web service. When working with Web services the namespaces that are required are summarized as follows:

System.Web.Services: Namespace consists a minimal and complete set of types needed to build a Web service
System.Web.Services.Description: This allows us to interact with WSDL programmatically
System.Web.Services.Discovery: These types allow a consumer to discover the Web services installed on a given machine
System.Web.Services.Protocols: This namespace defines a number of types that represents invocation protocols (HTTP-GET, HTTP-POST and SOAP)

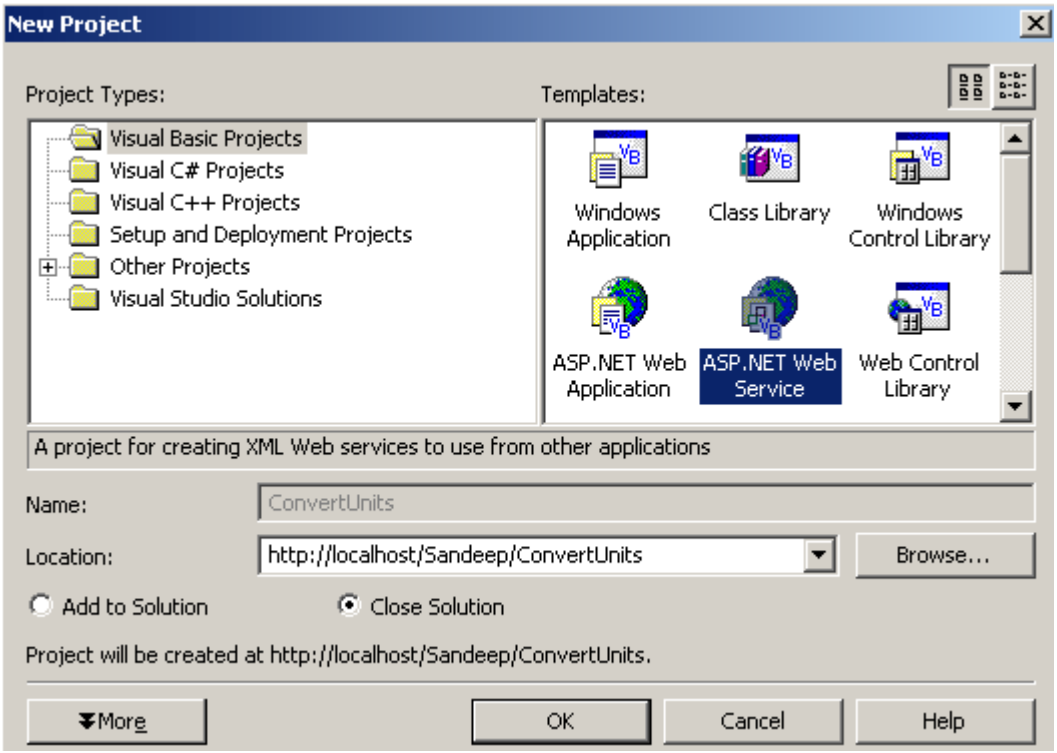
The System.Web.Services namespace

The System.Web.Services namespace is the namespace that we normally use in most of the projects as the types we need are already defined in this namespace. Following are the members of the System.Web.Services namespace:

WebMethodAttribute: Adding a <WebMethod()> attribute to a method in a Web service makes the method callable from a remote client through HTTP. This attribute exposes the functionality of the method to which it is applied to the outside world.
WebService: This defines the optional base class for Web Services.
WebServiceAttribute: The WebService attribute can be used to add information to a Web service that can describe it's functionality.
WebServiceBindingAttribute: Declares a binding protocol a given Web service method is implementing.

Coding a Sample Service

We will now create a sample service. This a simple service that converts a given distance from Kilometers to Miles and vice versa. Start Visual Studio .NET and open a new project from File->New-> Project. In the Projects Type pane select Visual Basic Projects and in the templates select ASP .NET Web Service, name this service as ConvertUnits and click OK. The new project dialog looks like the image below.



By default, Web service projects automatically create a new virtual directory under IIS and will store our files there. Switch to code view of the Web service to

take you to the code behind file which is a file with .asmx.vb extension. If you notice the Solution Explorer window you will find four files which are the Global.asax, Service1.asmx, ConvertUnits.vsdisco and the Web.config file. The Global.asax file allows us to respond to global-level events, the Web.config file allows us to declaratively configure our new Web service, the .asmx file is a Web service file that define the methods of the service and the .vsdisco file is a Discovery file that contains an XML description of the Web services at a given URL.

By default the code behind file looks like this when you open it:

```
Imports System.Web.Services
<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service2
Inherits System.Web.Services.WebService
#Region " Web Services Designer Generated Code "
' 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
' HelloWorld = "Hello World"
' End Function
End Class
```

We will build on the above mentioned code behind file. We will implement some simple functionality adding our own methods. The service which we will build will convert distance expressed in Kilometers to Miles and vice versa. The code for that looks like this:

```
Imports System
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1 Inherits
System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

#End Region
```

```
<WebMethod()> Public Function ConvertUnits(ByVal
EnterUnit As Decimal, _
                              ByVal FromUnits As String, ByVal
ToUnits As String)
'ConvertUnits function with three arguments
Select Case FromUnits.ToUpper.Chars(0)
'making a selection with Select Case
Case "K"
'for converting distance from kilometers to miles
Select Case ToUnits.ToUpper.Chars(0)
Case "K"
Return EnterUnit
'if both FromUnits and ToUnits are same, returns
the entered distance
Case "M"
Return EnterUnit / 1.4
'converts distance from kilometers to miles, assuming 1
mile=1.4 kilometer
Case Else
'to throw exception
End Select

Case "M"
'for converting distance from miles to kilometers
Select Case ToUnits.ToUpper.Chars(0)
Case "M"
Return EnterUnit
Case "K"
Return EnterUnit * 1.4
'converts distance from miles to kilometers
Case Else
'to throw exception
End Select
End Select
End Function

End Class
```

After finishing with the code run it by selecting Debug->Start from the main menu or by pressing F5 on the Keyboard. By default our browser functions as a makeshift client and shows an HTML view of the methods market with the <WebMethod()> attribute. Click here to view the page that first loads when you run this service. Click on the linkConvertUnits. Clicking on the link takes you to a page that provides Textbox types that allow us to enter some values in them. Enter some value in the EnterUnit field and in the FromUnits field enter either M

or K and in the ToUnits filed enter K or M. If you wish to convert 1000 Kilometers into Miles then you need to enter 1000 in the EnterUnit field, K in the FromUnits and M in the ToUnits. Once you are done with it, click invoke. This will invoke the method we wrote in code and the result will be returned via an XML attribute. Click here to run the service now. That's all it takes to create a simple Web service.

Sample Service 2

In this section we will create a Calculator Web service that works similar to a Calculator and performs operations like Add, Subtract, Multiply, Divide and we will consume this Web service with a Visual Basic Windows Application. To start, open a new project and select ASP .NET Web service or add a new Web service to the existing ASP .NET Web service project by right-clicking the project name in Solution Explorer and selecting Add->Add Web service. Name this project as Calculator, open the code behind file and start writing the following code.

```
Imports System
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

#End Region

<WebMethod(Description:="Click to Add numbers")> Public Function
Add_
(ByVal x As Integer, ByVal y As Integer) As Integer
'this method adds two numbers by accepting the input from the user
'Description property allows to document the functionality of the Web
method.
Return x + y
End Function

<WebMethod(Description:="Click to Subtract numbers")> Public
Function Subtract_
(ByVal x As Integer,ByVal y As Integer) As Integer
'this method subtracts by accepting the input from the user
Return x - y
End Function
```

```
<WebMethod(Description:="Click to Multiply numbers")> Public
Function Multiply_
(ByVal x As Integer,ByVal y As Integer) As Integer
'this method multiplies two numbers by accepting the input from the user
Return x * y
End Function

<WebMethod(Description:="Click to Divide numbers")> Public Function
Divide_
(ByVal x As Integer,ByVal y As Integer) As Integer
'this method divides two numbers by accepting the input from the user
If (y = 0) Then
Throw New Exception("Can't divide by zero")
'if number entered is 0 throws an exception
End If
Return x / y
End Function

End Class
```

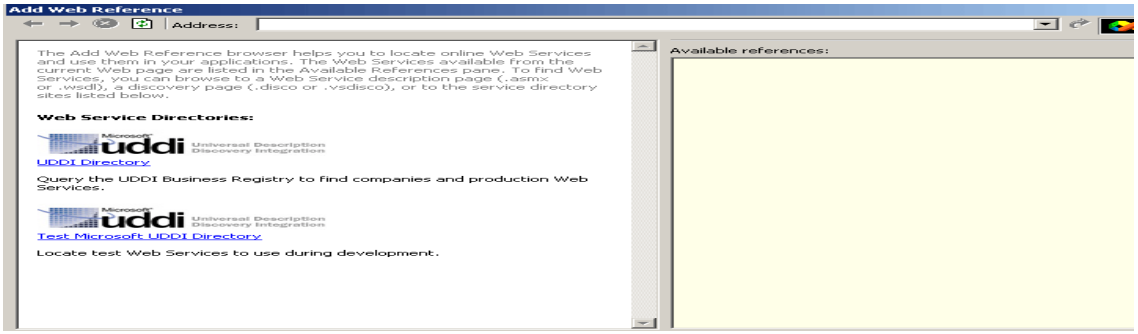
Once when you finish with the code run the service by selecting Debug->Start from the main menu or by pressing F5 on the keyboard. The Service that loads can be viewed by clicking here. You can view all the methods we created in code along with the method description on that page. Also you can enter some values in the Textboxes and test the service. We will consume this service in a Windows Form.

Consuming this Web service

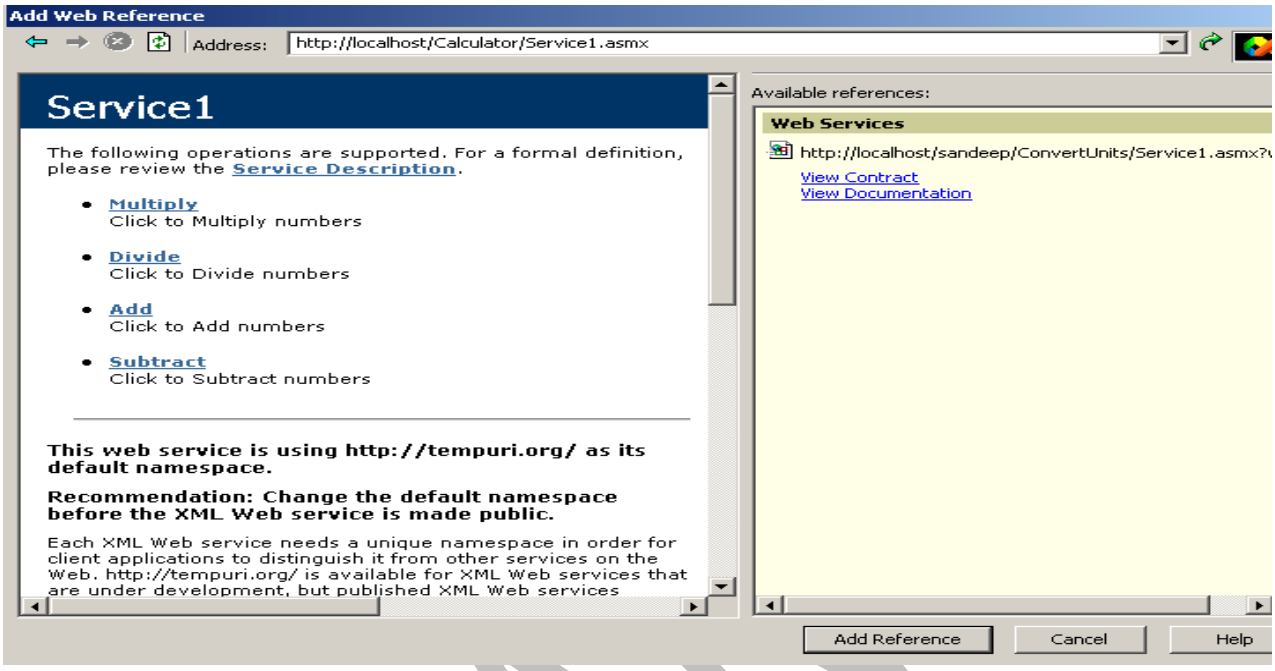
Open a new Visual Basic Project and select Windows Application from the template. From the toolbox add a Button to the form. Our intention here is to consume the Web service which we created with this Windows Application. When we click the Button it will call the method which we specify in it's click event and will return the calculated result ina MessageBox.

Adding Web Service Reference to the Windows Application

We can add a reference to the Web service in two ways, with Solution Explorer and using WSDL tool. In the Solution Explorer right click on references and select Add Web Reference. That opens up a template similar to the image below.



In the address bar type the URL of the Calculator service which we created. Since it's in the root directory of IIS you need to type the following address:**http://localhost/Calculator/Service1.asmx**. It should look like the image below.



After the Calculator service is loaded, click Add Reference. That adds a reference to the Calculator service.

To use WSDL tool to add a reference to this Web service, open Visual Studio .NET command prompt, change the folder in the command prompt to the location where you created Calculator and type the following:
WSDL "**http://localhost/Calculator/Service1.asmx**" **/l:VB**. After you finish typing the command, in Solution Explorer, right-click Calculator, select Add and then click Add Existing Item. Locate Service1.vb, and then click to select it. Click Open.

Calling the Service from Windows Form

Open Form1 and place the following code. Recall that we are calling a method when the Button in this application is clicked. We need to create an instance of the proxy classlocalhost.Service1 and call the function, passing a string argument. The code for that looks like this:

```
Public Class Form1
Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

#End Region

Dim myService As localhost.Service1 = New localhost.Service1()
'creating an instance
```

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles MyBase.Load

End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
MessageBox.Show("Sum is " & myService.Add(10, 20))
'calling the Add method in the Web Service returning the result in a
messagebox
End Sub

End Class
```

Once you finish with the application, run the form and click on the Button. The sum of two numbers will be displayed in a MessageBox. We not only created a Web service but also consumed the service in other application.

Sample Service 3

In this section we will build a more interesting Web service that returns a ADO .NET DataSet, containing the full set of records from a table. We will create our own databasetable and access the data from the table with this Web service. To Start, open Microsoft Access and create a new Database named Currency. Create a new table Table1 and add three columns named, Country Code, Country Name and Currency. Enter some values in the table and close it. Open Visual Studio .NET and select ASP .NET Web servicefrom the projects type template. Drag a OleDb connection from the Data tab in the toolbox and using the properties window build a connection string that connects to the Currency database which we created. Switch to the code view and start writing the following code.

```
Imports System
Imports System.Web.Services
Imports System.Data.OleDb
'import this namespace as we are working with an OleDb source

<WebService(Namespace := "http://tempuri.org/")> _

Public Class Service1 Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

#End Region

<WebMethod()> Public Function GetData() As DataSet
'WebMethod name is GetData,generate data set
Dim da as OleDbDataAdapter=new OleDbDataAdapter("Select * From
```

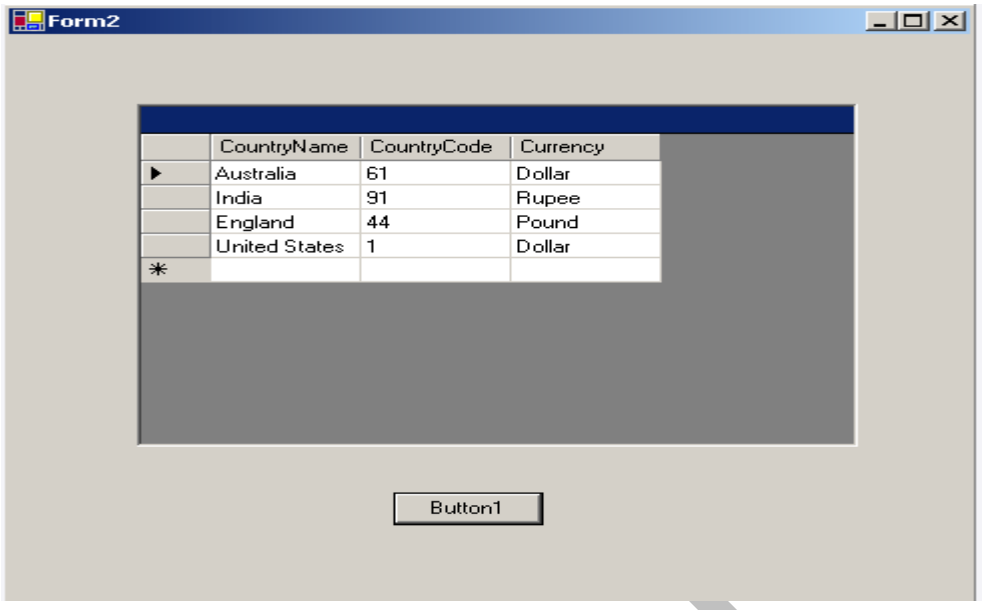
```
Table1",_
OleDbConnection1)
'dataadapter
Dim ds As DataSet=new DataSet()
'declaring a new DataSet
da.Fill(ds, "Table1")
'filling dataadapter
Return ds
'returning dataset
End Function
End Class
```

Consuming the Service

Once you finish with coding the Web service we need to consume this service. To do that, open a new Windows Application and from the toolbox drag a DataGrid and a Button. Our intention here is to load the data from Table1 in the Currency database into the DataGrid when we click the Button. Now, add a Web reference to the Web serviceby selecting Reference->Add WebReference in the Solution Explorer Window. Enter the URL of the service in the address bar and click "Add Reference". That adds a reference to the Web Service. Now double-click on the Button and write the following code.

```
Public Class Form1 Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

#End Region

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs)Handles Button1.Click
Dim myService As New localhost.Service1()
'an instance of the Web service
Dim ds1 As DataSet = myService.GetData
DataGrid1.DataSource = ds1.Tables("Table1")
'filling the datagrid with table
End Sub
End Class
```

Once you finish with the code, run the Windows Application and click on the Button. The data you entered in Table1 of the Currency database will be displayed in the datagrid. The difference, we are accessing the data with a Web service. The image below displays that.



Deploying XML Web Services

Deploying a Web Service is enabling a Web service to execute on a specific Web server. Before deploying a Web service the first thing you need to do is change the namespace of the Web service and make sure it specifies a unique namespace. The reason why you need to change the namespace is to avoid conflicts with other Web services. The default Web service namespace is set to "http://tempuri.org" and there exists a possibility that others might be using the same namespace. The namespace you specify is used within the WSDL document of the Web service to uniquely identify the callable entry points of the service. It is recommended that you specify a namespace URI that you own or have under you control. Using your domain name as part of Web service namespace gurantee uniqueness. ASP .NET Web services support aNamespace property as part of the WebService attribute which is set to tempuri.org by default. The following code sample shows how to set a namespace to a Web service.

```
Imports System
Imports System.Web.Services

&ltWebService(Namespace := "http://startvbdotnet.com/namespaces/") />
_
Public Class Service1 Inherits System.Web.Services.WebService

Implementing Code

End Class
```

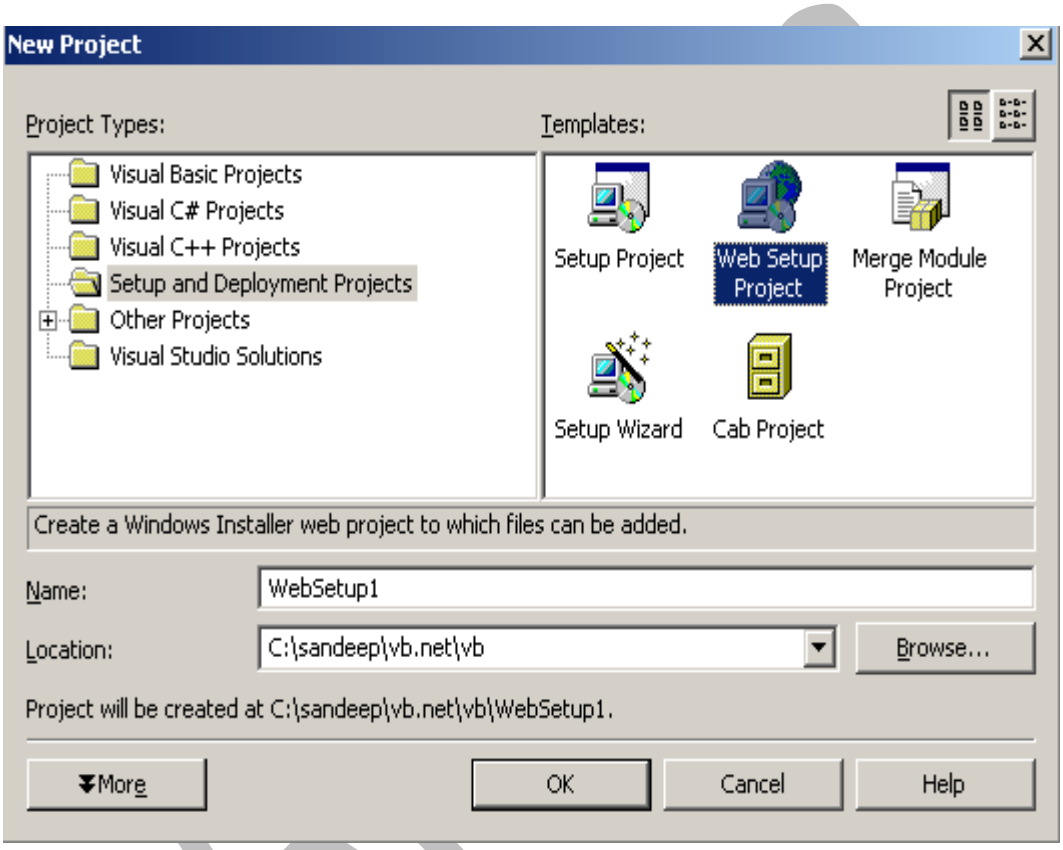
Deploying Web services

In general, deploying a Web service is copying the Web service entry point file (ASMX file), the Web service assembly and related assemblies and other support files like Web.config, etc, to the target Web server. Some Web services may just require you to copy the ASMX file

on to the target Web server. The tools you can use to deploy a Web service are: VS .NET Web Setup Project and VS .NET Project Copy.

VS .NET Web Setup Project

If you build your Web services with Visual Studio .NET then you can use the Web Setup Project wizard to deploy your Web service. The Web Setup project creates a MSI file that when executed, creates and configures a virtual directory on the Web server, copies all the required files to execute the Web service and registers any additional assemblies needed by the Web service. The image below displays the new project dialogue with Web Setup as the selection.



The steps required to deploy a Web service using the Web Setup project are as follows:

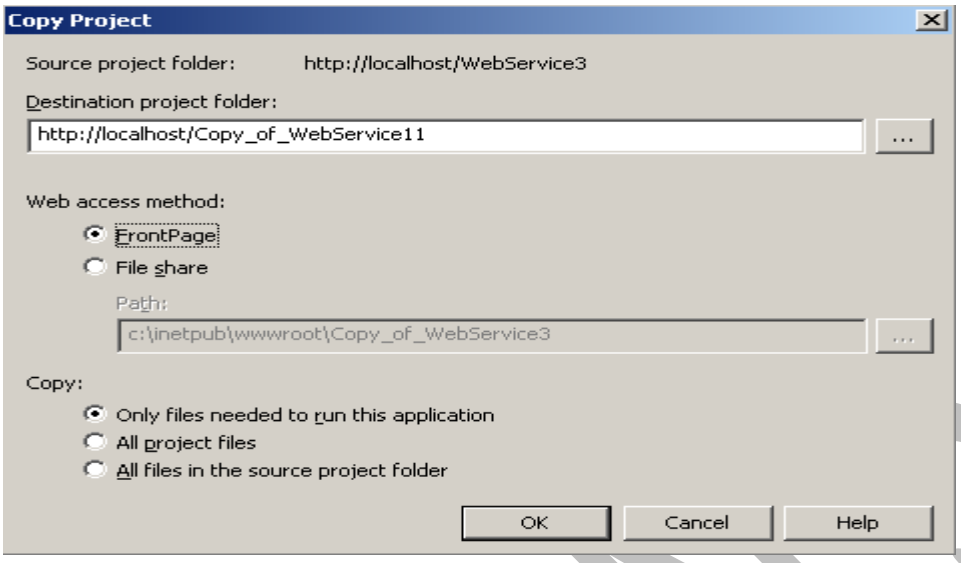
1. Create the Web Setup project using the Web Setup Project template in VS .NET.
2. Build the project.
3. Copy the installation package to the target Web server
4. Run the installation package on the Web server.

When you use Web Setup Project to deploy your Web services you have an option to specify an alternate virtual directory during the setup process and also the setupprocess creates a new virtual directory and configures the virtual directory for the Web service.

VS .NET Project Copy

VS .NET project copy is another method for deploying Web services. This is a simple process for deployment but it does not perform the tasks like configuring a virtual

directory and file registrations that the Web service may require. You can view the Project Copy dialogue by selecting Project->Copy Project from main menu. Tha image below displays that.



As you can see from the above image, you have three options while using this method. They are:

Only files needed to run this application: Copies all DLLs with references in the bin folder as well as any files marked with a BuildAction of Content.
All project files: Copies all project files created and managed by VS .NET.
All files in the source project folder: Copies all VS .NET project files as well as other files that reside in the project folders.

Publishing and Security

Publishing a Web service means enabling a Web service user (consumer) to locate the service description and instructing the consumer how they should interact with the Web service. The process of locating and interrogating Web service description is called the discovery process. There are two ways for the discovery of Web services, DISCOand UDDI.

DISCO

We use DISCO if the number of consumers using our service are relatively small. We can directly give them the path of our Web Server and deploy the DISCO file on the Web Server. When we build a Web service, Visual Studio automatically creates a DISCO file. This file has an extension of .vsdisco and is stored in the virtual directory of IIS along with the asmx file. This DISCO file contains links to resources that describe the Web service.

Creating a Proxy using wsdl.exe

If we want consumers to program against our Web Service, we have to create a proxy and an assembly. We can generate the proxy using the WebServiceUtil.exe command-line tool with Visual Studio .NET command prompt. The wsdl.exe command line tool generates a code file that represents the proxy to the remote Web service. We need to specify the name of

the proxy file to be generated and the URL where the WSDL can be obtained. The command for that is:
wsdl.exe /l:VB /out:c:\convertproxy.vb http://localhost/ConvertUnits/service1.asmx?WSDL. This line illustrates the use of wsdl tool to generate a proxy class for our ConvertUnits Web service. The wsdl.exe utility generates C# (C-Sharp) code by default. If we want our proxy written in VB .NET we can use the optional /l:(language) as we did in the command line.

Creating a Proxy Using Visual Studio

We can also use Visual Studio to create the proxy class. We did that in the Sample Service 2 section. Visual Studio automatically creates a Web service proxy classes using the Add Web Reference feature. All we have to do is provide the location of the WSDL document for the Web service and Visual Studio takes care of the rest.

UDDI

The Universal Discovery, Description, and Integration (UDDI) project provides a global directory of Web Services. UDDI enables consumers to search and locate Web services if the consumer is not aware of the exact location of the service or the owner of the service. UDDI is for Web services like Google is for Web pages. UDDI allows us to easily find Web services based on a centralized and globally available registry of businesses which are accessible over the Internet. If you have a Web service and if you wish to publish it with UDDI then you need to visit the UDDI web site and register your service there.

Finding Services

UDDI directory allows us to search for companies providing services. All we need to do is visit the UDDI web site and search for the service we are interested in. Web sites like Google, Amazon and EBay are also providing their services through their web sites. You can visit their sites and download the SDK. The SDK provides all the information you need to access their Web services along with the documentation that helps you in it's implementation.

Security Configuration

We can us the Config.web file for all security related configuration as all the information is in this file. We have the ability to configure three fundamental functions forsecurity: authentication, authorization, and impersonation. The Config.web will have three additional sequences enclosed in the parent <security> tag.

Authentication, Authorization, Impersonation

All your Web clients communicate with your Web application through IIS. So you can use IIS authentication (Basic, Digest, and NTLM/Kerberos) in addition to the ASP.NET built-in authentication solutions. Some ASP .NET authentication providers are:

-Passport authentication, which is a centralized authentication service provided by Microsoft.
-Cookie authentication, which issues a cookie to the request/response that contains the

credentials for reacquiring the identity.
-Windows authentication, which is used in conjunction with IIS authentication.

IIS authentication methods assume that the user is already known to the server, while ASP .NET methods don't. With Passport authentication your site has to support Microsoft Passport credentials, and Cookie authentication assigns an identity to an "unknown stranger" who complies with some rules. Once a client request is authenticated and an identity is given, we have to determine whether this identity is allowed to have access to the requested resource.

ASP .NET distinguishes two modes of authorization: file and URL. File authorization is active when using Windows authentication. To determine if access should be granted or not, a check against an Access Control List (ACL) is done. In URL authorization, identities are mapped to pieces of the Uniform Resource Identifier (URI) namespace to selectively allow access to parts of the namespace.

When using impersonation, IIS and Windows file access security play a role. IIS authenticates the user using Basic, Digest, or Windows NTLM/Kerberos authentication. IIS then passes a token to ASP .NET, the token is either authenticated or unauthenticated.

Code Access Security

Apart from the ASP .NET built-in security features, we can make use of code access security feature of the .NET Framework. With code access security we can admit code originating from one computer system to be executed safely on another system. Therefore the code's identity and origin has to be verified. To determine whether the code should be authorized or not, the runtime's security system walks the call stack, checking for each caller whether access to a resource or performing an operation should be allowed. In the .NET Framework you must specify the operations the code is allowed to perform. This can be done, in the assembly of your Web application.

Extensible Markup Language (XML)

The markup language most widely used today is undoubtedly Hyper Text Markup Language (HTML), which is used to create Webpages. A Markup language describes the structure of the document. HTML is based on Standard Generalized Markup Language (SGML), which is an application of SGML. Webpages designed using HTML are designed using HTML predefined tags. These days, as Internet is used widely as general form of communication and as transferring data over the Internet is becoming more intensive and handling that data more complex many Web Developers are turning to XML as their alternative to HTML. It's worth having a brief overview of this wonderful new Markup Language which is changing the way data is handled on the Internet.

What is XML?

XML is a meta-markup language which means that it lets us create our own markup language (our own tags).

XML is popular for the following reasons:

- It Allows Easy Data Exchange

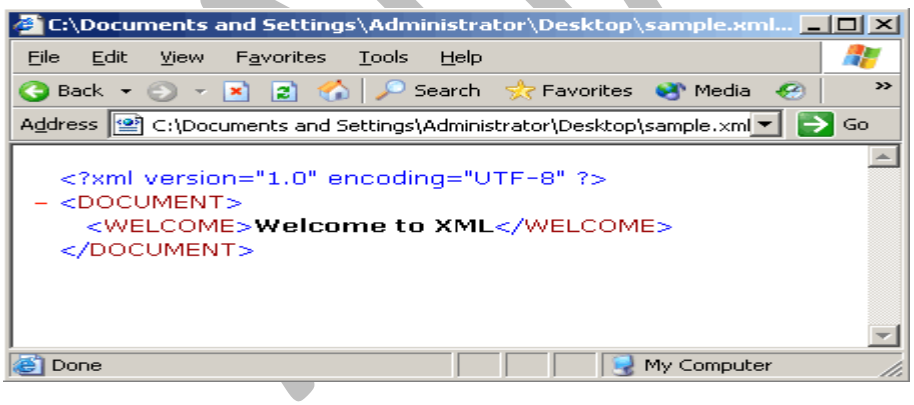
- It Allows to Customize Markup languages
- Makes the data in the document Self-Describing
- Allows for Structured and Integrated data

The current version of XML is 1.0 and XML is case sensitive. Let's follow this meta-markup language with an example. Save the following code with a .xml extension.

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
<WELCOME>
Welcome to XML
</WELCOME>
</DOCUMENT>>
```

Breaking the above code for understanding:

- The document starts with the XML processing instruction <?xml version="1.0" encoding="UTF-8"?>
- All XML processing instructions should start and end with ?
- xml version="1.0" means the version of XML, which is currently 1.0
- UTF-8 is a 8-bit condensed version of Unicode
- The document starts with the <DOCUMENT> element which may or may not contain other elements within it and should always end with </DOCUMENT>. All other elements should be between <DOCUMENT> and </DOCUMENT> making <DOCUMENT> the root element for this XML page.
- The next element is <WELCOME> between the <DOCUMENT> and </DOCUMENT> and which contains a message, Welcome to XML.The above code when opened in a browser looks like the image below.

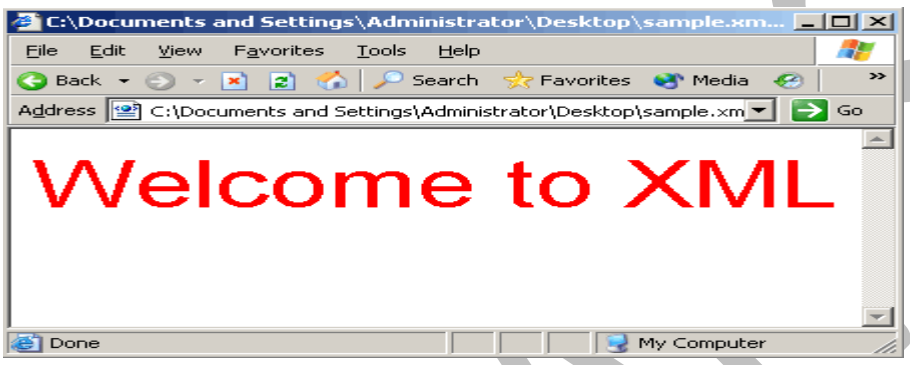


To format the content of the elements created in the document we use a style sheet to tell the browser the way the document should be. Alternatively, programming languages like Java and JavaScript can be used. Lets take a look how the above example looks when formatted using style sheet.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="style.css"?>
<DOCUMENT>
<WELCOME>
Welcome to XML
```

```
</WELCOME>
</DOCUMENT>
```

The above code includes a new line <?xml-stylesheet type="text/css" href="style.css"?> which means that the type of style sheet being used is CSS (Cascading Style Sheet, XSL can also be used) and it's name is style.css.
The file style.css looks like this: WELCOME{font-size:40pt;font-family:Arial; color:red}
This file states that it's customizing the <WELCOME> element to display it's content in a 40 pt font with arial as it's font and it's color as red.
You can customize different elements to display their content in different fonts and colors.
Make sure that the file style.css is saved in the same directory where the xml file is saved.
The output after adding the style sheet looks like the image below.



XML is case sensitive, which means <WeLCOME> and </Welcome> are treated differently. <WELCOME> should be closed with a corresponding</WELCOME> tag.

Well-Formed XML Documents

If an XML document is not understood successfully by an XML processor then the processor cannot format the document. To handle that, XML documents are subject to two constraints: well formedness and validity, well formedness being the basic constraint.

Well-Formed Document

As set by the W3C, for an XML document to be well formed it should follow the document production containing three parts in the document.

- A prolog
- A root element
- Optional miscellaneous part

The prolog should include an XML declaration such as <?xml version="1.0"?>. It can also contain a Document Type Definition (DTD).
The root element of a document can hold other elements and the document should contain exactly one root element. All other elements should be enclosed within the root element.
The optional miscellaneous part can be made up of XML comments, processing instructions and whitespaces.
Also the XML document should follow the syntax rules specified in the XML 1.0 recommendation set by W3C.

An example of a well formed document is listed below :

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
<CONSUMER>
<NAME>
<FIRST_NAME>
BEN
</FIRST_NAME>
<LAST_NAME>
HOLLIAKE
</LAST_NAME>
</NAME>
<PURCHASE>
<ORDER>
<ITEM>
DVD
</ITEM>
<QUANTITY>
1
</QUANTITY>
<PRICE>
200
</PRICE>
</ORDER>
</PURCHASE>
</CONSUMER>
<CONSUMER>
<NAME>
<FIRST_NAME>
ADAM
</FIRST_NAME>
<LAST_NAME>
ANDERSON
</LAST_NAME>
</NAME>
<PURCHASE>
<ORDER>
<ITEM>
VCR
</ITEM>
<QUANTITY>
1
</QUANTITY>
<PRICE>
150
</PRICE>
</ORDER>
</PURCHASE>
</CONSUMER>
```

```
</DOCUMENT>
```

Understanding the above document for well-formedness:

- The document starts with a prolog, which is the xml declaration.
- The First element, which is the root element is the <DOCUMENT> element which contains all other elements.
- Next is the <CONSUMER> element inside the root element which is for two consumers.
- For each consumer, their name is stored in the <NAME> element which itself contains elements like <FIRST_NAME> and <LAST_NAME>.
- The details of the purchases which the consumer made is stored in the <ORDER> element in the <PURCHASE> element which in turn contains the elements <ITEM><QUANTITY><PRICE> which records the item purchased, quantity and price which the consumer purchased.
- The document ends with the closing </DOCUMENT> element.

Data can be stored for as many consumers as wanted and handling such kind of data is not a problem for the XML processor.

The following are the basic rules that should be kept on mind when creating a Well-Formed XML document.

- The document should start with an XML declaration
- The document should be included with one or more elements
- For elements that are not empty include start and end tags
- All elements of the document should be contained within the root element
- Elements should be nested correctly

Documents like the one above can be extended as long as we can. XML doesn't have any problem handling such kind of documents, as long as they are wellformed.

Valid XML Documents

An XML document is said to be valid if it has a Document Type Definition (DTD) or XML schema associated with it and if the document complies with it. DTD's are all aboutspecifying the structure of the document and not the content of the document. And with a common DTD many XML applications can be shared. Such is the importance of a DTD.

Let's take a look at the example which was created in the section Well-Formed XML documents.

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
<CONSUMER>
<NAME>
<FIRST_NAME>
BEN
</FIRST_NAME>
```

```
<LAST_NAME>
HOLLIAKE
</LAST_NAME>
</NAME>
<PURCHASE>
<ORDER>
<ITEM>
DVD
</ITEM>
<QUANTITY>
1
</QUANTITY>
<PRICE>
200
</PRICE>
</ORDER>
</PURCHASE>
</CONSUMER>
<CONSUMER>
<NAME>
<FIRST_NAME>
ADAM
</FIRST_NAME>
<LAST_NAME>
ANDERSON
</LAST_NAME>
</NAME>
<PURCHASE>
<ORDER>
<ITEM>
VCR
</ITEM>
<QUANTITY>
1
</QUANTITY>
<PRICE>
150
</PRICE>
</ORDER>
</PURCHASE>
</CONSUMER>
</DOCUMENT>
```

Adding a DTD to the example above makes the code look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DOCUMENT[
<!ELEMENT DOCUMENT (CONSUMER)*>
<!ELEMENT CONSUMER (NAME,PURCHASE)>
```

```
<!ELEMENT NAME (FIRST_NAME,LAST_NAME)>
<!ELEMENT FIRST_NAME (#PCDATA)>
<!ELEMENT LAST_NAME (#PCDATA)>
<!ELEMENT PURCHASE (ORDER)*>
<!ELEMENT ORDER (ITEM,QUANTITY,PRICE)>
<!ELEMENT ITEM (#PCDATA)>
<!ELEMENT QUANTITY (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
]>
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
<CONSUMER>
<NAME>
<FIRST_NAME>
BEN
</FIRST_NAME>
<LAST_NAME>
HOLLIAKE
</LAST_NAME>
</NAME>
<PURCHASE>
<ORDER>
<ITEM>
DVD
</ITEM>
<QUANTITY>
1
</QUANTITY>
<PRICE>
200
</PRICE>
</ORDER>
</PURCHASE>
</CONSUMER>
<CONSUMER>
<NAME>
<FIRST_NAME>
ADAM
</FIRST_NAME>
<LAST_NAME>
ANDERSON
</LAST_NAME>
</NAME>
<PURCHASE>
<ORDER>
<ITEM>
VCR
</ITEM>
<QUANTITY>
1
```

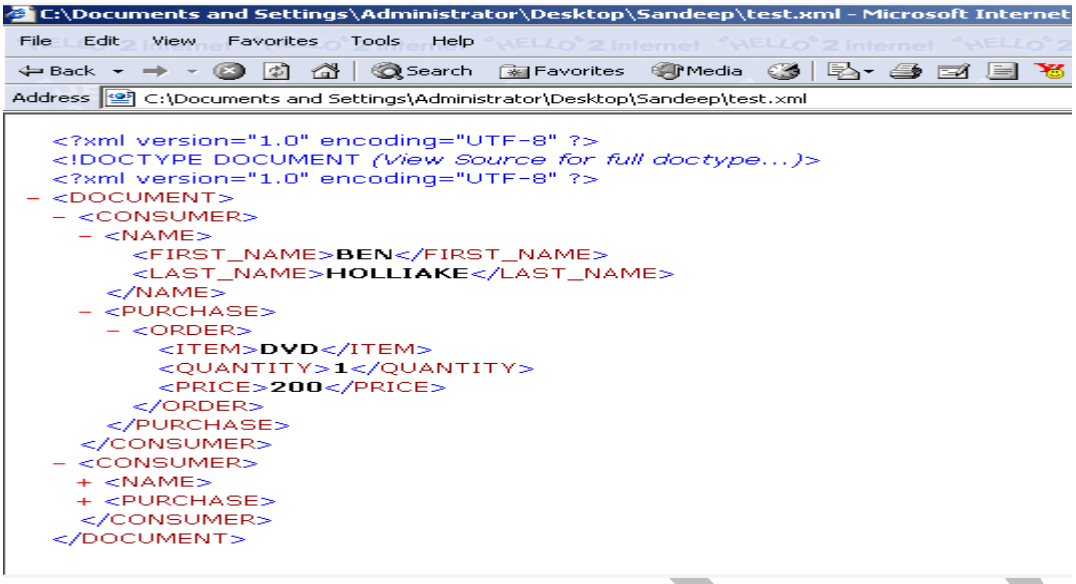
```
</QUANTITY>
<PRICE>
150
</PRICE>
</ORDER>
</PURCHASE>
</CONSUMER>
</DOCUMENT>
```

Breaking the DTD for understanding:

- Note the first line of the DTD, <!DOCTYPE DOCUMENT[. That line is the document type declaration.<!DOCTYPE> is the syntax to declare a DTD and it should be followed by the root element, which in this example is the DOCUMENT element.
- Each element should be specified with the syntax <!ELEMENT>. Using that declaration we can specify whether each element is a parsed character data (#PCDATA, used for storing plain text) or can contain other elements in it.
- In the example above the CONSUMER element is written like this <!ELEMENT DOCUMENT(CONSUMER)*>.The asterik(*) here indicates that the CONSUMER element can have zero or more occurrences. In the example above, it has two occurrences.
- The next element in the CONSUMER element is the NAME element which in turn contains the elements FIRST_NAME and LAST_NAME within it.
- Both the FIRST_NAME and LAST_NAME elements are declared as #PCDATA which allows them to handle plain text.
- The next element in the DTD is the PURCHASE element with an asterik(*) which means that it has zero or more occurrences.
- The elements within the PURCHASE element is the ORDER element which in turn include the elements ITEM, QUANTITY and PRICE.
- The elements ITEM, QUANTITY and PRICE are declared as #PCDATA as they hold only plain text.

That's how a basic DTD looks like. A DTD like the one above is said to be an internal DTD. We can also create external DTD's and it's these external DTD's which allows us to share a common XML document within different organizations.

For more information about how to insert attributes, comments, etc in DTD's please refer to the W3C specification for XML DTD's. The image below shows how the above code when opened in an browser looks like.



Controls

A control is an object that can be drawn on to the Form to enable or enhance user interaction with the application. Examples of these controls, TextBoxes, Buttons, Labels, Radio Buttons, etc. All these Windows Controls are based on the Control class, the base class for all controls. Visual Basic allows us to work with controls in two ways: atdesign time and at runtime. Working with controls at design time means, controls are visible to us and we can work with them by dragging and dropping them from the Toolbox and setting their properties in the properties window. Working at runtime means, controls are not visible while designing, are created and assigned properties in code and are visible only when the application is executed. There are many new controls added in Visual Basic .NET and we will be working with some of the most popular controls in this section. You can select the controls from the menu towards the left-hand side of this page.

Notable properties of most of these Windows Controls which are based on the Control class itself are summarized in the table below. You can always find the properties of the control with which you are working by pressing F4 on the keyboard or by selecting View->Properties Window from the main menu.

The Control Class

The Control class is in the System.Windows.Forms namespace. It is a base class for the Windows Controls. The class hierarchy is shown below.

Object
 MarshalByRefObject
  Component
   Control
    ButtonBase, Etc, Etc
     Button, Etc, Etc

Main class is the Object class from which MarshalByRefObject class is derived and the Component class is derived from the MarshalByRefObject class and so on.

The properties of the Control object are summarized below. Properties are alphabetical as seen in the properties window.

Property	Description
AllowDrop	Indicates if the form can accept data that the user drags and drops into it
Anchor	Gets/Sets which edges of the control are anchored
BackColor	Gets/Sets the background color for the control
BackgroundImage	Gets/Sets the background image in the control
Bottom	Gets the distance bewteen the bottom of the control and the top of its container client area
Bounds	Gets/Sets the controls bounding rectangle
CanFocus	Returns a value specifying if the control can recieve focus
CanSelect	Returns a value specifying if the control can be selected
Capture	Gets/Sets a value specifying if the control has captured the mouse
CausesValidation	Gets/Sets a value specfying if the control causes validation for all controls that require validation
ContainsFocus	Returns a value specifying if the control has the input focus
ContextMenu	Gets/Sets the shortcut menu for the control
Controls	Gets/Sets the collection of controls contained within the control
Cursor	Gets/Sets the cursor to be displayed when the user moves the mouse over the form
DataBindings	Gets the data bindings for the control
Dock	Gets/Sets which edge of the parent a control is docked to
Enabled	Gets/Sets a value indicating if the control is enabled

Focused	Returns a value specifying if the control has input focus
Font	Gets/Sets the font for the control
ForeColor	Gets/Sets the foreground color of the control
HasChildren	Returns a value specifying if the control contains child controls
Height	Gets/Sets the height of the control
Left	Gets/Sets the x-coordinates of a control's left edge in pixels
Location	Gets/Sets the co-ordinates of the upper-left corner of the control
Name	Gets/Sets name for the control
Parent	Gets/Sets the control's parent container
Right	Returns the distance between the right edge of the control and the left edge of it's container
RightToLeft	Gets/Sets the value indicating if the alignment of the control's elements is reversed to supportright-to-left fonts
Size	Gets/Sets size of the control in pixels
TabIndex	Gets/Sets the tab order of this control in its container
TabStop	Gets/Sets a value specifying if the user can tab to this control with the tab key
Tag	Gets/Sets an object that contains data about the control
Text	Gets/Sets the text for this control
Top	Gets/Sets the top coordinates of the control
Visible	Gets/Sets a value specifying if the control is visible
Width	Gets/Sets the width of the control

Control Tab Order

To move focus from one control to other quickly using the keyboard we can use the Tab key. We can set the order in which the focus is transferred by setting the tab order. The tab order is the order in which controls on the form receive the focus and is specified by the TabIndex property. To change the order in which a control receives focus we need to set the TabIndex property to different value for each control on the form. Lower values receive the focus first and proceed numerically through higher values. If there is a tie between TabIndex values, the focus first goes to the control that is closest to the front of the form.

We can also set the tab order graphically with Visual Studio by selecting Tab Index from the View menu. Boxes containing current tab order appear in each control when you select Tab Index from View menu. Click each control to set the correct tab order in which you want the controls to receive focus.

Button Control

One of the most popular control in Visual Basic is the Button Control (previously Command Control). They are the controls which we click and release to perform some action. Buttons are used mostly for handling events in code, say, for sending data entered in the form to the database and so on. The default event of the Button is the Click event and the Button class is based on the ButtonBase class which is based on the Control class.

Button Event

The default event of the Button is the Click event. When a Button is clicked it responds with the Click Event. The Click event of Button looks like this in code:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
'You place the code here to perform action when Button is clicked
End Sub
```

Working with Buttons

Well, it's time to work with Buttons. Drag a Button from the toolbox onto the Form. The default text on the Button is Button1. Click on Button1 and select it's properties bypressing F4 on the keyboard or by selecting View->Properties Window from the main menu. That displays the Properties for Button1.

Important Properties of Button1 from Properties Window:

Appearance

Appearance section of the properties window allows us to make changes to the appearance of the Button. With the help of BackColor and Background Image properties we can set a background color and a background image to the button. We set the font color and font style for the text that appears on button with ForeColor and the Fontproperty. We change the appearance style of the button with the FlatStyle property. We can change the text that

appears on button with the Text property and with theTextAlign property we can set where on the button the text should appear from a predefined set of options.

Behavior

Notable Behavior properties of the Button are the Enabled and Visible properties. The Enabled property is set to True by default which makes the button enabled and setting it's property to False makes the button Disabled. With the Visible property we can make the Button Visible or Invisible. The default value is set to True and to make the button Invisible set it's property to False.

Layout

Layout properties are about the look of the Button. Note the Dock property here. A control can be docked to one edge of its parent container or can be docked to all edges and fill the parent container. The default value is set to none. If you want to dock the control towards the left, right, top, bottom and center you can do that by selecting from the button like image this property displays. With the Location property you can change the location of the button. With the Size property you can set the size of the button. Apart from the Dock property you can set it's size and location by moving and stretching the Button on the form itself.

Below is the image of a Button.

Creating a Button in Code

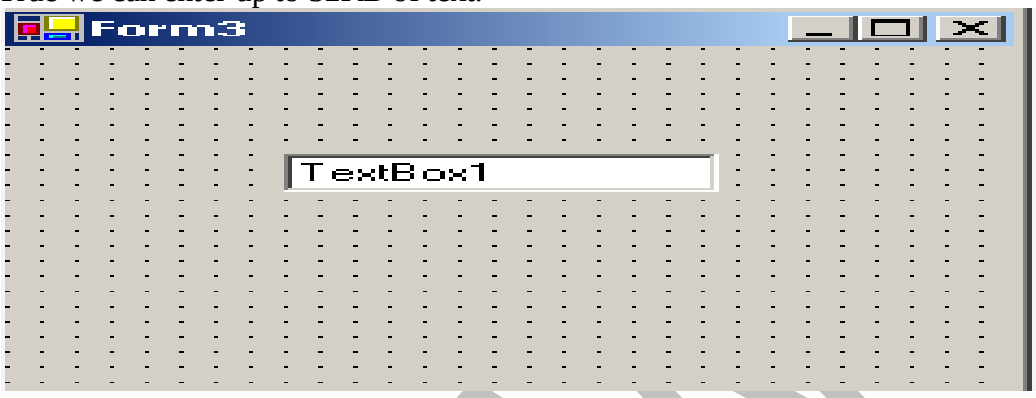
Below is the code to create a button.

```
Public Class Form1 Inherits System.Windows.Forms.Form
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles_ MyBase.Load
Dim Button1 as New Button()
'declaring the button, Button1
Button1.Text="Creating a Button"
'setting the text to be displayed on the Button
Button1.Location=New Point(100,50)
'setting the location for the Button where it should be created
Button1.Size=New Size(75,23)
'setting the size of the Button
Me.Controls.Add(Button1)
'adding the Button that is created to the form
'the Me keyword is used to refer to the current object, in this case the Form
```

```
End Sub
End Class
```

TextBox Control

Windows users should be familiar with textboxes. This control looks like a box and accepts input from the user. The TextBox is based on the TextBoxBase class which is based on the Control class. TextBoxes are used to accept input from the user or used to display text. By default we can enter up to 2048 characters in a TextBox but if theMultiline property is set to True we can enter up to 32KB of text.



The image below displays a Textbox.

Some Notable Properties:

Some important properties in the Behavior section of the Properties Window for TextBoxes.

Enabled: Default value is True. To disable, set the property to False.
Multiline: Setting this property to True makes the TextBox multiline which allows to accept multiple lines of text. Default value is False.
PasswordChar: Used to set the password character. The text displayed in the TextBox will be the character set by the user. Say, if you enter *, the text that is entered in the TextBox is displayed as *.
ReadOnly: Makes this TextBox readonly. It doesn't allow to enter any text.
Visible: Default value is True. To hide it set the property to False.

Important properties in the Appearance section

TextAlign: Allows to align the text from three possible options. The default value is left and you can set the alignment of text to right or center.
Scrollbars: Allows to add a scrollbar to a Textbox. Very useful when the TextBox is multiline. You have four options with this property. Options are are None, Horizontal, Vertical and Both. Depending on the size of the TextBox anyone of those can be used.

TextBox Event

The default event of the TextBox is the TextChanged Event which looks like this in code:

```
Private Sub TextBox1_TextChanged(ByVal sender As System.Object,
```

```
ByVal e As _
System.EventArgs) Handles TextBox1.TextChanged

End Sub
```

Working With TextBoxes

Lets work with some examples to understand TextBoxes.

Drag two TextBoxes (TextBox1, TextBox2) and a Button (Button1) from the toolbox.

Code to Display some text in the TextBox

We want to display some text, say, "Welcome to TextBoxes", in TextBox1 when the Button is clicked. The code looks like this:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
TextBox1.Text = "Welcome to TextBoxes"
End Sub
```

Code to Work with PassWord Character

Set the PasswordChar property of TextBox2 to *. Setting that will make the text entered in TextBox2 to be displayed as *. We want to display what is entered in TextBox2 in TextBox1. The code for that looks like this:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
TextBox1.Text = TextBox2.Text
End Sub
```

When you run the program and enter some text in TextBox2, text will be displayed as *. When you click the Button, the text you entered in TextBox2 will be displayed as plain text in TextBox1.

Code to Validate User Input

We can make sure that a TextBox can accept only characters or numbers which can restrict accidental operations. For example, adding two numbers of the form 27+2J cannot return anything. To avoid such kind of operations we use the KeyPress event of the TextBox.

Code that allows you to enter only double digits in a TextBox looks like this:

```
Private Sub TextBox1_KeyPress(ByVal sender As Object,ByVal e As_
System.Windows.Forms.KeyPressEventArgs) Handles
TextBox1.KeyPress
If(e.KeyChar < "10" Or e.KeyChar > "100") Then
```

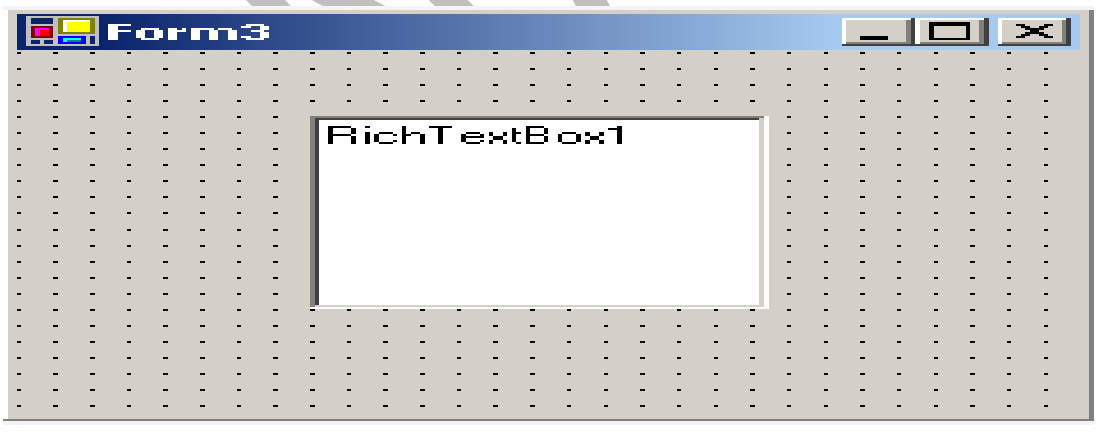
```
MessageBox.Show("Enter Double Digits")
End If
End Sub
```

Creating a TextBox in Code

```
Public Class Form1 Inherits System.Windows.Forms.Form
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles MyBase.Load
Dim TextBox1 as New TextBox()
TextBox1.Text="Hello Mate"
TextBox1.Location=New Point(100,50)
TextBox1.Size=New Size(75,23)
Me.Controls.Add(TextBox1)
End Sub
End Class
```

RichTextBox

RichTextBoxes are similar to TextBoxes but they provide some advanced features over the standard TextBox. RichTextBox allows formatting the text,
say adding colors,displaying particular font types and so on. The RichTextBox, like the TextBox is based on the TextBoxBase class which is based on the Control class. These RichTextBoxes came into existence because many word processors these days allow us to save text in a rich text format. With RichTextBoxes we can also create our own word processors. We have two options when accessing text in a RichTextBox, text and rtf (rich text format). Text holds text in normal text and rtf holds text in rich text format. Image of a RichTextBox is shown below.



RichTextBox Event

The default event of RichtextBox is the TextChanged event which looks like this in code:

```
Private Sub RichTextBox1_TextChanged(ByVal sender As
System.Object, _
ByVal e As System.EventArgs) Handles RichTextBox1.TextChanged
```

```
End Sub
```

Code Samples

Code for creating bold and italic text in a RichTextBox

Drag a RichTextBox (RichTextBox1) and a Button (Button1) onto the form. Enter some text in RichTextBox1, say, "We are working with RichTextBoxes". Paste the following code in the click event of Button1. The following code will search for text we mention in code and sets it to be displayed as Bold or Italic based on what text is searched for.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
RichTextBox1.SelectionStart = RichTextBox1.Find("are")
'using the Find method to find the text "are" and setting it's
'return property to SelectionStart which selects the text to format
Dim ifont As New Font(RichTextBox1.Font, FontStyle.Italic)
'creating a new font object to set the font style
RichTextBox1.SelectionFont = ifont
'assigning the value selected from the RichTextBox the font style
RichTextBox1.SelectionStart = RichTextBox1.Find("working")
Dim bfont As New Font(RichTextBox1.Font, FontStyle.Bold)
RichTextBox1.SelectionFont = bfont
End Sub
```

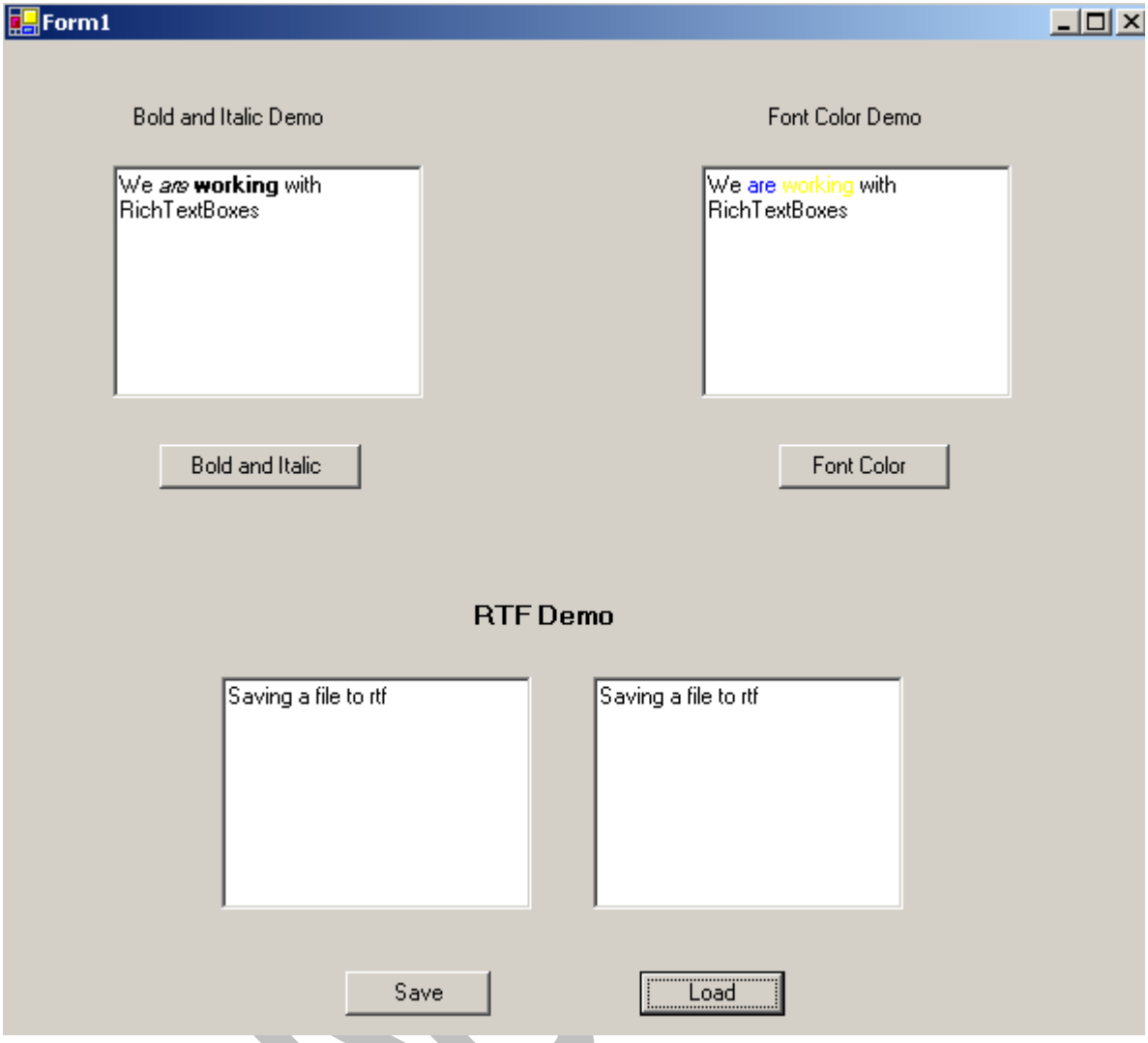
When you run the above code and click Button1, the text "*are*" is displayed in Italic and the text "**working**" is displayed in Bold font. The image below displays the output.

Code for Setting the Color of Text

Lets work with previous example. Code for setting the color for particular text looks like this:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles Button1.Click
RichTextBox1.SelectionStart = RichTextBox1.Find("are")
'using the Find method to find the text "are" and setting it's return
'property to SelectionStart which selects the text
RichTextBox1.SelectionColor = Color.Blue
'setting the color for the selected text with SelectionColor property
RichTextBox1.SelectionStart = RichTextBox1.Find("working")
RichTextBox1.SelectionColor = Color.Yellow
End Sub
```

The output when the Button is Clicked is the text "are" being displayed in Blue and the text "working" in yellow as shown in the image below.



Code for Saving Files to RTF

Drag two RichTextBoxes and two Buttons (Save, Load) onto the form. When you enter some text in RichTextBox1 and click on Save button, the text from RichTextBox1 is saved into a rtf (rich text format) file. When you click on Load button the text from the rtf file is displayed into RichTextBox2. The code for that looks like this:

```
Private Sub Save_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Save.Click
RichTextBox1.SaveFile("hello.rtf")
'using SaveFile method to save text in a rich text box to hard disk
End Sub

Private Sub Load_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Load.Click
RichTextBox2.LoadFile("hello.rtf")
'using LoadFile method to read the saved file
End Sub
```

The files which we create using the SaveFile method are saved in the bin directory of the Windows Application. You can view output of the above said code in the image above.

Label, LinkLabel

Label

Labels are those controls that are used to display text in other parts of the application. They are based on the Control class.

Notable property of the label control is the text property which is used to set the text for the label.

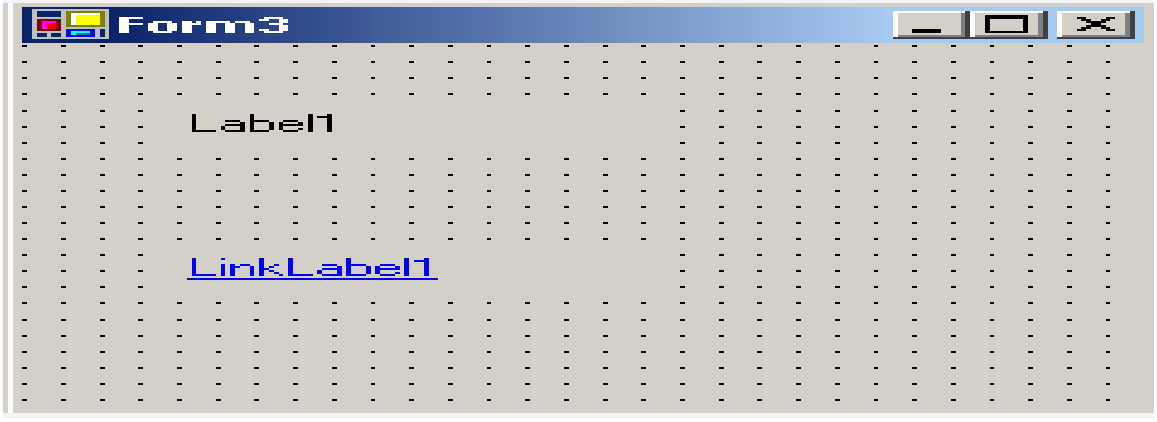
Label Event

The default event of Label is the Click event which looks like this in code:

```
Private Sub Label1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)_
Handles Label1.Click

End Sub
```

Creating a Label in Code

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e
As System.EventArgs)_
Handles MyBase.Load Dim Label1 As New Label()
Label1.Text = "Label"
Label1.Location = New Point(135, 70)
Label1.Size = New Size(30, 30)
Me.Controls.Add(Label1)
End Sub
```



LinkLabel

LinkLabel is similar to a Label but they display a hyperlink. Even multiple hyperlinks can be specified in the text of the control and each hyperlink can perform a different task within the application. They are based on the Label class which is based on the Control class.

Notable properties of the LinkLabel control are
the ActiveLinkColor, LinkColor and LinkVisited which are used to set the link color.

LinkLabel Event

The default event of LinkLabel is the LinkClicked event which looks like this in code:

```
Private Sub LinkLabel1_LinkClicked(ByVal sender As System.Object, _
ByVal e As System.Windows.Forms.LinkLabelLinkClickedEventArgs)_
Handles LinkLabel1.LinkClicked

End Sub
```

Working with LinkLabel

Drag a LinkLabel (LinkLabel1) onto the form. When we click this LinkLabel it will take us to "**www.startvbdotnet.com**". The code for that looks like this:

```
Private Sub LinkLabel1_LinkClicked(ByVal sender As System.Object,
ByVal_
e As System.Windows.Forms.LinkLabelLinkClickedEventArgs)_
Handles LinkLabel1.LinkClicked
System.Diagnostics.Process.Start("www.startvbdotnet.com")
'using the start method of system.diagnostics.process class
'process class gives access to local and remote processes
End Sub
```

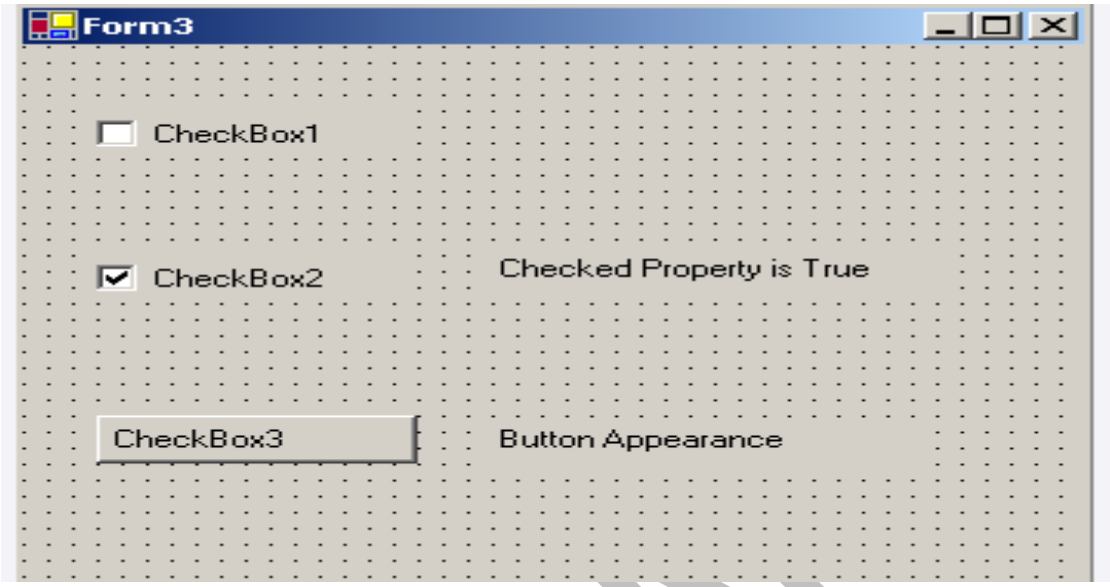
Creating a LinkLabel in Code

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e
As System.EventArgs)_
Handles MyBase.Load
Dim LinkLabel1 As New LinkLabel()
LinkLabel1.Text = "Label"
LinkLabel1.Location = New Point(135, 70)
LinkLabel1.Size = New Size(30, 30)
Me.Controls.Add(LinkLabel1)
End Sub
```

CheckBox

CheckBoxes are those controls which gives us an option to select, say, Yes/No or True/False. A checkbox is clicked to select and clicked again to deselect some option. When

a checkbox is selected a check (a tick mark) appears indicating a selection.
The CheckBox control is based on the TextBoxBase class which is based on the Controlclass.
Below is the image of a Checkbox.



Notable Properties

Important properties of the CheckBox in the Appearance section of the properties window are:

Appearance: Default value is Normal. Set the value to Button if you want the CheckBox to be displayed as a Button.
BackgroundImage: Used to set a background image for the checkbox.
CheckAlign: Used to set the alignment for the CheckBox from a predefined list.
Checked: Default value is False, set it to True if you want the CheckBox to be displayed as checked.
CheckState: Default value is Unchecked. Set it to True if you want a check to appear. When set to Indeterminate it displays a check in gray background.
FlatStyle: Default value is Standard. Select the value from a predefined list to set the style of the checkbox.

Important property in the Behavior section of the properties window is the ThreeState property which is set to False by default. Set it to True to specify if the Checkbox can allow three check states than two.

CheckBox Event

The default event of the CheckBox is the CheckedChange event which looks like this in code:

```
Private Sub CheckBox1_CheckedChanged(ByVal sender As
System.Object, _
ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
```

```
End Sub
```

Working with CheckBoxes

Lets work with an example. Drag a CheckBox (CheckBox1), TextBox (TextBox1) and a Button (Button1) from the Toolbox.

Code to display some text when the Checkbox is checked

```
Private Sub CheckBox1_CheckedChanged(ByVal sender As
System.Object, _
ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
TextBox1.Text = "CheckBox Checked"
End Sub
```

Code to check a CheckBox's state

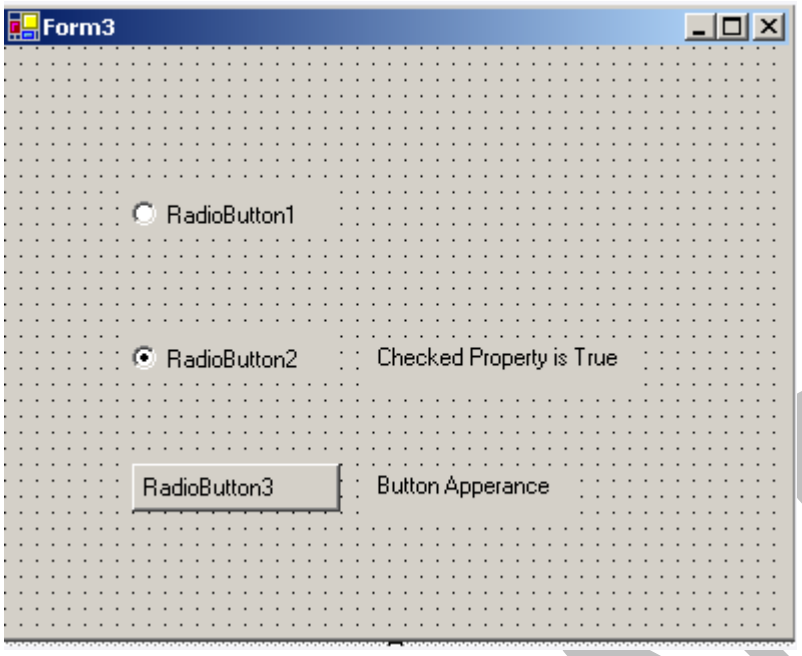
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
If CheckBox1.Checked = True Then
TextBox1.Text = "Checked"
Else
TextBox1.Text = "UnChecked"
End If
End Sub
```

Creating a CheckBox in Code

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles_ MyBase.Load
Dim CheckBox1 As New CheckBox()
CheckBox1.Text = "Checkbox1"
CheckBox1.Location = New Point(100, 50)
CheckBox1.Size = New Size(95, 45)
Me.Controls.Add(CheckBox1)
End Sub
```

RadioButton

RadioButtons are similar to CheckBoxes but RadioButtons are displayed as rounded instead of boxed as with a checkbox. Like CheckBoxes, RadioButtons are used to select and deselect options but they allow us to choose from mutually exclusive options. The RadioButton control is based on the ButtonBase class which is based on the Controlclass. A major difference between CheckBoxes and RadioButtons is, RadioButtons are mostly used together in a group. Below is the image of a RadioButton.



Important properties of the RadioButton in the Appearance section of the properties window are:

Appearance: Default value is Normal. Set the value to Button if you want the RadioButton to be displayed as a Button.
BackgroundImage: Used to set a background image for the RadioButton.
CheckAlign: Used to set the alignment for the RadioButton from a predefined list.
Checked: Default value is False, set it to True if you want the RadioButton to be displayed as checked.
FlatStyle: Default value is Standard. Select the value from a predefined list to set the style of the RadioButton.

RadioButton Event

The default event of the RadioButton is the CheckedChange event which looks like this in code:

```
Private Sub RadioButton1_CheckedChanged(ByVal sender As
System.Object,_
ByVal e As System.EventArgs) Handles RadioButton1.CheckedChanged

End Sub
```

Working with Examples

Drag a RadioButton (RadioButton1), TextBox (TextBox1) and a Button (Button1) from the Toolbox.

Code to display some text when the RadioButton is selected

```
Private Sub RadioButton1_CheckedChanged(ByVal sender As
System.Object, _
ByVal e As System.EventArgs) Handles RadioButton1.CheckedChanged
TextBox1.Text = "RadioButton Selected"
End Sub
```

Code to check a RadioButton's state

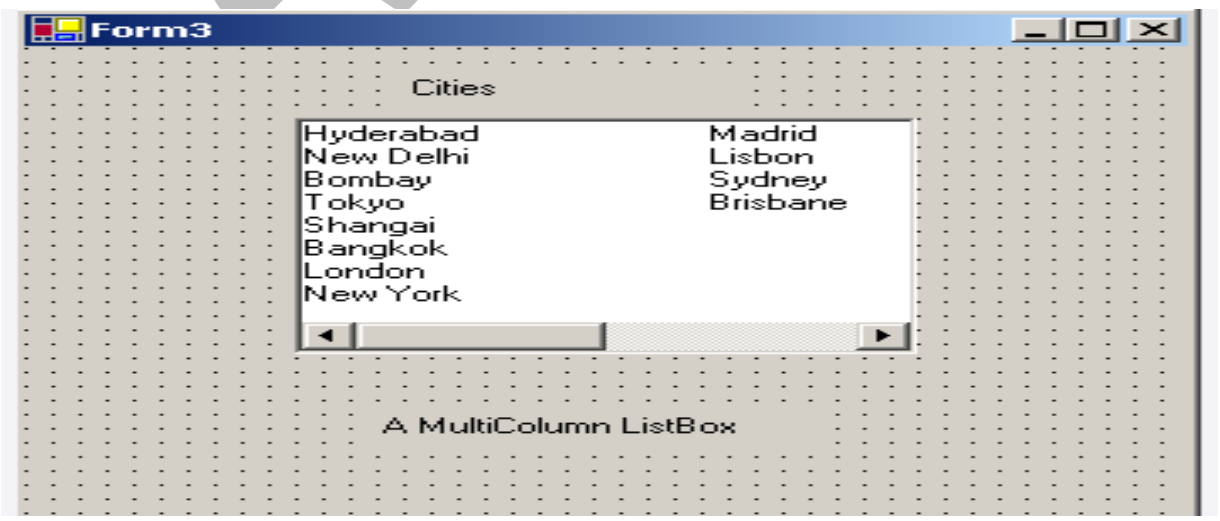
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
If RadioButton1.Checked = True Then
TextBox1.Text = "Selected"
Else
TextBox1.Text = "Not Selected"
End If
End Sub
```

Creating a RadioButton in Code

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles_ MyBase.Load
Dim RadioButton1 As New RadioButton()
RadioButton1.Text = "RadioButton1"
RadioButton1.Location = New Point(120,60)
RadioButton1.Size = New Size(100, 50)
Me.Controls.Add(RadioButton1)
End Sub
```

ListBox

The ListBox control displays a list of items from which we can make a selection. We can select one or more than one of the items from the list. The ListBox control is based on the ListControl class which is based on the Control class. The image below displays a ListBox.



Notable Properties of the ListBox

In the Behavior Section

HorizontalScrollbar: Displays a horizontal scrollbar to the ListBox. Works when the ListBox has MultipleColumns.
MultiColumn: The default value is set to False. Set it to True if you want the list box to display multiple columns.
ScrollAlwaysVisible: Default value is set to False. Setting it to True will display both Vertical and Horizontal scrollbar always.
SelectionMode: Default value is set to one. Select option None if you do not any item to be selected. Select it to MultiSimple if you want multiple items to be selected. Setting it to MultiExtended allows you to select multiple items with the help of Shift, Control and arrow keys on the keyboard.
Sorted: Default value is set to False. Set it to True if you want the items displayed in the ListBox to be sorted by alphabetical order.

In the Data Section

Notable property in the Data section of the Properties window is the Items property. The Items property allows us to add the items we want to be displayed in the list box. Doing so is simple, click on the ellipses to open the String Collection Editor window and start entering what you want to be displayed in the ListBox. After entering the items click OK and doing that adds all the items to the ListBox.

ListBox Event

The default event of ListBox is the SelectedIndexChanged which looks like this in code:

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As
System.Object, _
ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged

End Sub
```

Working with ListBoxes

Drag a TextBox and a ListBox control to the form and add some items to the ListBox with it's items property.

Referring to Items in the ListBox

Items in a ListBox are referred by index. When items are added to the ListBox they are assigned an index. The first item in the ListBox always has an index of 0 the next 1 and so on.

Code to display the index of an item

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As
```

```
System.Object, _
ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged
TextBox1.Text = ListBox1.SelectedIndex
'using the selected index property of the list box to select the index
End Sub
```

When you run the code and select an item from the ListBox, it's index is displayed in the textbox.

Counting the number of Items in a ListBox

Add a Button to the form and place the following code in it's click event.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles Button1.Click
TextBox1.Text = ListBox1.Items.Count
'counting the number of items in the ListBox with the Items.Count
End Sub
```

When you run the code and click the Button it will display the number of items available in the ListBox.

Code to display the item selected from ListBox in a TextBox

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As
System.Object,_
ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged
TextBox1.Text = ListBox1.SelectedItem
'using the selected item property
End Sub
```

When you run the code and click an item in the ListBox that item will be displayed in the TextBox.

Code to Remove items from a ListBox

You can remove all items or one particular item from the list box.

Code to remove a particular item

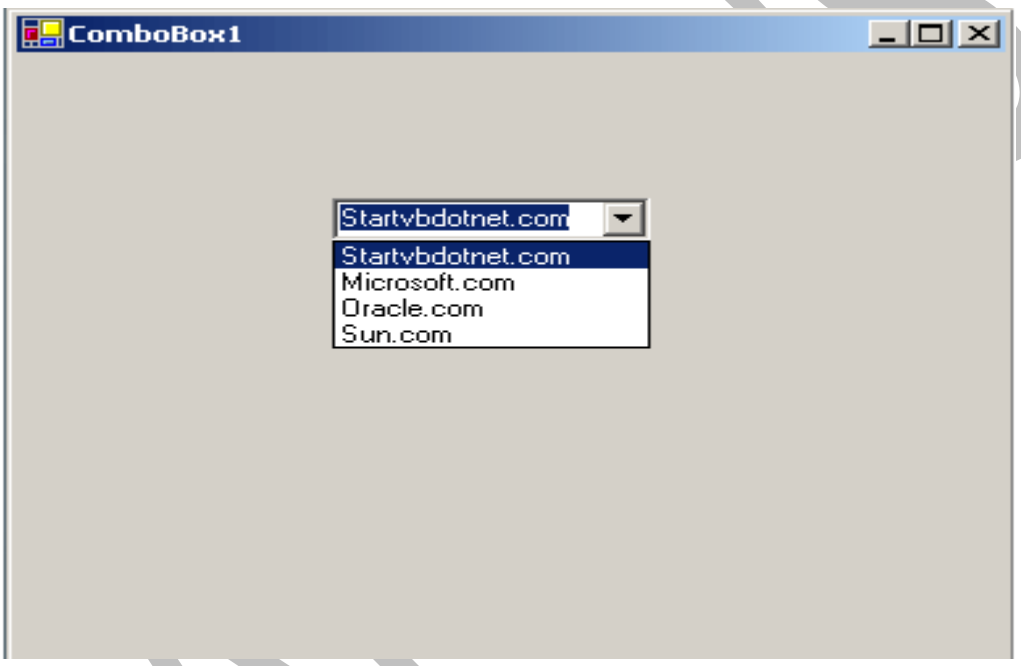
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles Button1.Click
ListBox1.Items.RemoveAt(4)
'removing an item by specifying it's index
End Sub
```

Code to Remove all items

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
ListBox1.Items.Clear()
'using the clear method to clear the list box
End Sub
```

ComboBox

ComboBox is a combination of a TextBox and a ListBox. The ComboBox displays an editing field (TextBox) combined with a ListBox allowing us to select from the list or to enter new text. ComboBox displays data in a drop-down style format. The ComboBox class is derived from the ListBox class. Below is the Image of a ComboBox.



Notable properties of the ComboBox

The DropDownStyle property in the Appearance section of the properties window allows us to set the look of the ComboBox. The default value is set to DropDown which means that the ComboBox displays the Text set by it's Text property in the Textbox and displays it's items in the DropDownListBox below. Setting it to simple makes theComboBox to be displayed with a TextBox and the list box which doesn't drop down. Setting it to DropDownList makes the ComboBox to make selection only from the drop down list and restricts you from entering any text in the textbox.

We can sort the ComboBox with it's Sorted property which is set to False by Default.

We can add items to the ComboBox with it's Items property.

ComboBox Event

The default event of ComboBox is SelectedIndexChanged which looks like this in code:

```
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As
System.Object,_
ByVal e As System.EventArgs) Handles
ComboBox1.SelectedIndexChanged

End Sub
```

Working with ComboBoxes

Drag a ComboBox and a TextBox control onto the form. To display the selection made in the ComboBox in the Textbox the code looks like this:

```
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As
System.Object,_
ByVal e As System.EventArgs) Handles
ComboBox1.SelectedIndexChanged
TextBox1.Text = ComboBox1.SelectedItem
'selecting the item from the ComboBox with selected item property
End Sub
```

Removing items from a ComboBox

You can remove all items or one particular item from the list box part of the ComboxBox. Code to remove a particular item by it's Index number looks like this:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
ComboBox1.Items.RemoveAt(4)
'removing an item by specifying it's index
End Sub
```

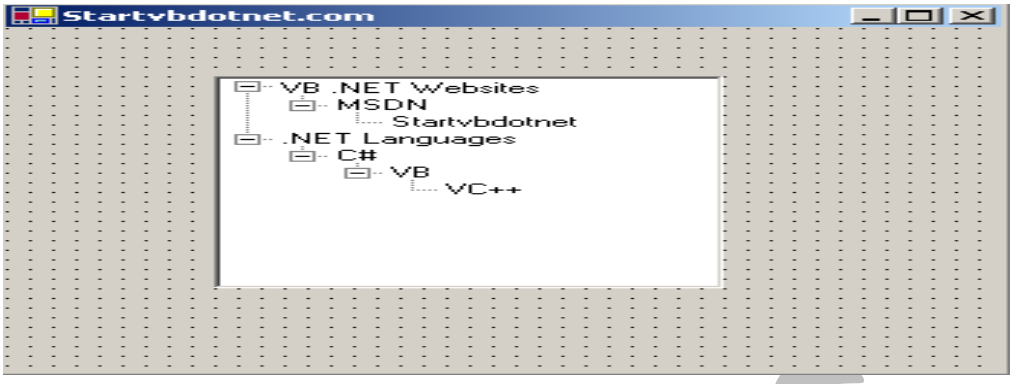
Code to remove all items from the ComboBox

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
ComboBox1.Items.Clear()
'using the clear method to clear the list box
End Sub
```

TreeView

The tree view control is used to display a hierarchy of nodes (both parent, child). You can expand and collpase these nodes by clicking them. This control is similar to Windows Explorer which displays a tree view in it's left pane to list all the folders on the hard

disk. Below is the image of a Tree View control.



Notable Properties of TreeView

Bounds: Gets the actual bound of the tree node
Checked: Gets/Sets whether the tree node is checked
FirstNode: Gets the first child tree node
FullPath: Gets the path from the root node to the current node
ImageIndex: Gets/Sets the image list index of the image displayed for a node
Index: Gets the location of the node in the node collection
IsEditing: Gets whether the node can be edited
IsExpaned: Gets whether the node is expaned
IsSelected: Gets whether the node is selected
LastNode: Gets the last child node
NextNode: Gets the next sibling node
NextVisibleNode: Gets the next visible node
NodeFont: Gets/Sets the font for nodes
Nodes: Gets the collection of nodes in the current node
Parent: Gets the parent node of the current node
PrevNode: Gets the previous sibling node
PrevVisibleNode: Gets the previous visible node
TreeView: Gets the node's parent tree view

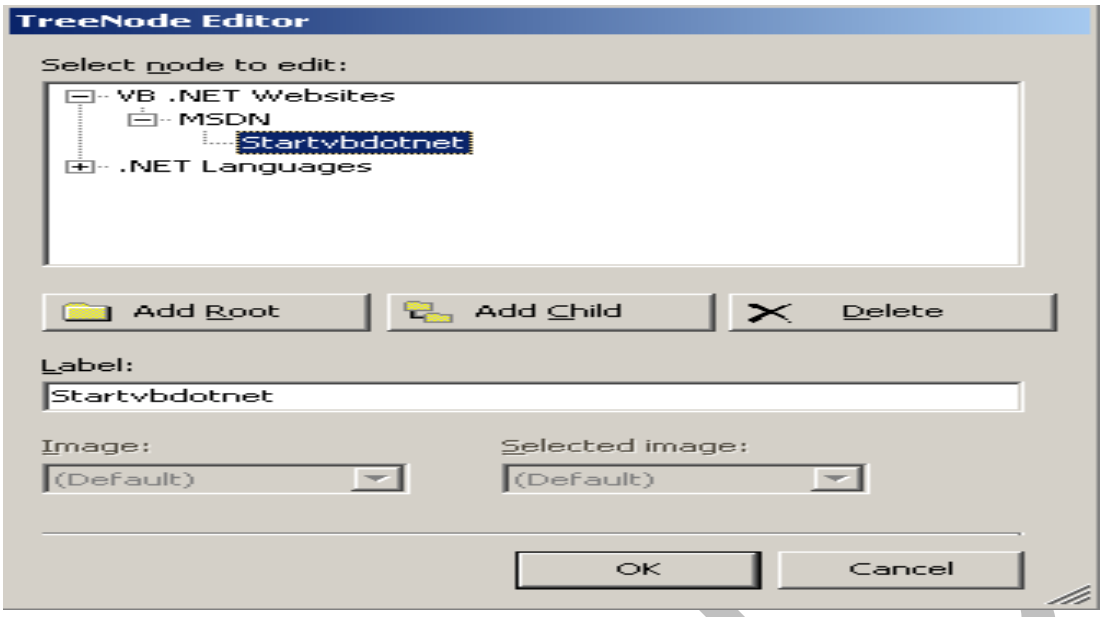
TreeView Event

Default event of the Tree View control is the AfterSelect event which looks like this in code:

```
Private Sub TreeView1_AfterSelect(ByVal sender As System.Object,
ByVal e As_
System.Windows.Forms.TreeViewEventArgs) Handles
TreeView1.AfterSelect

End Sub
```

Working with Tree Views

Drag a Tree View control on to a form and to add nodes to it select the nodes property in the properties window, which displays the TreeNode editor as shown below.

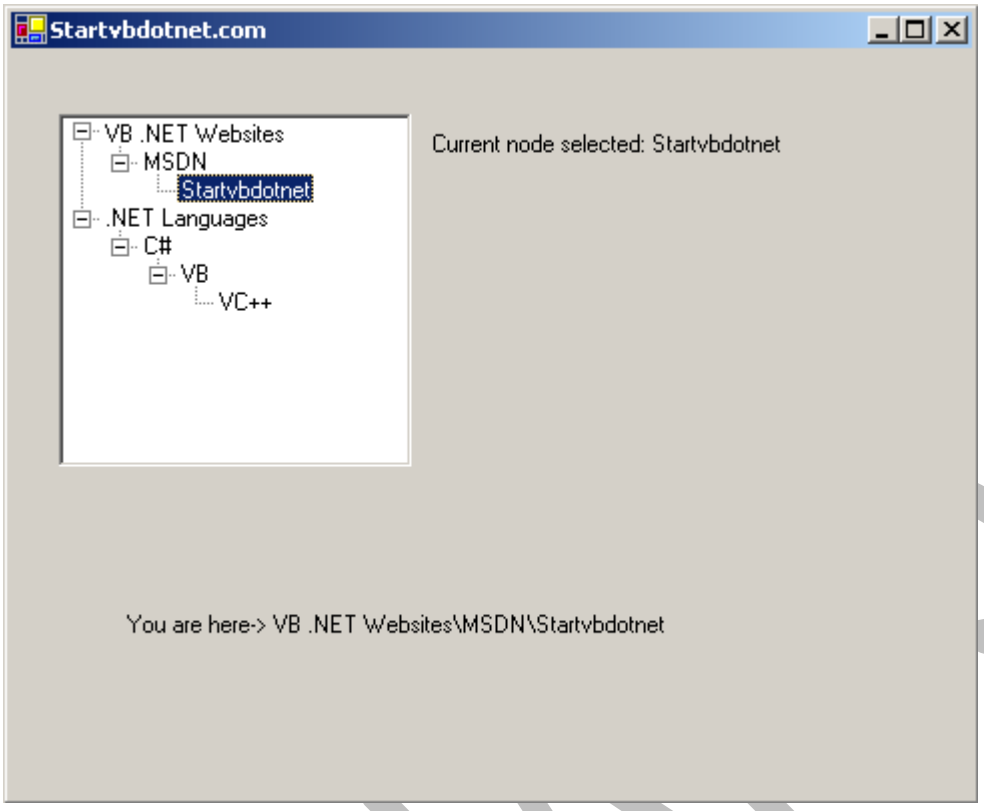


To start adding nodes, you should click the Add Root button, which adds a top-level node. To add child nodes to that node, you should select that node and use the Add Child button. To set text for a node, select the node and set it's text in the textbox as shown in the image above.

Assuming you added some nodes to the tree view, drag two Labels (Label1, Label2) from the toolbox on to the form. The following code displays the node you select on Label2 and the path to that node on Label1. The code looks like this:

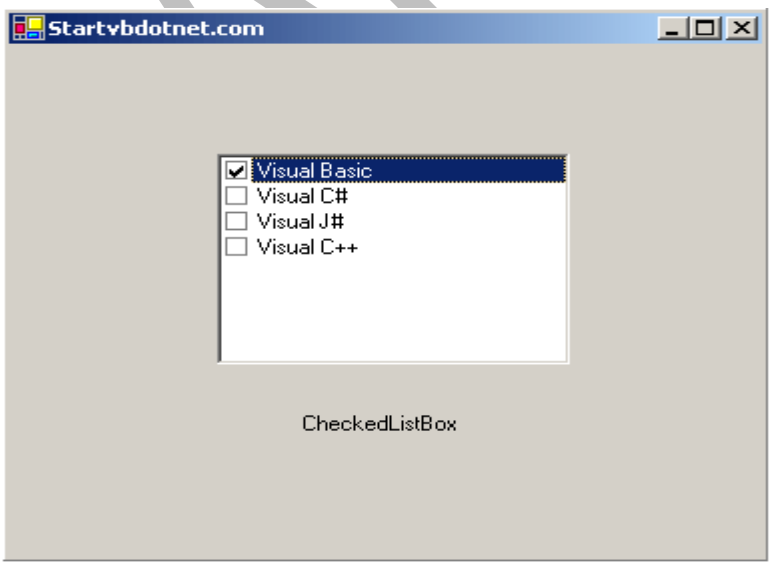
```
Public Class Form12 Inherits System.Windows.Forms.Form
#Region " Windows Form Designer generated code "

#End Region

Private Sub TreeView1_AfterSelect(ByVal sender As System.Object,
ByVal e As_
System.Windows.Forms.TreeViewEventArgs) Handles
TreeView1.AfterSelect
Label1.Text = "You are here->" & " " & e.Node.FullPath
'displaying the path of the selected node
Label2.Text = "Current node selected:" & " " & e.Node.Text
'displaying the selected node
End Sub

End Class
```

The image below displays sample output from above code.



CheckedListBox

As the name says, CheckedListBox is a combination of a ListBox and a CheckBox.
It displays a ListBox with a CheckBox towards it's left. The CheckedListBox class is derived from the ListBox class and is based on that class. Since the CheckedListBox is derived from the ListBox it shares all the members of ListBox. Below is the Image of a CheckedListBox.



Notable Properties of CheckedListBox

The notable property in the appearance section of the properties window is the ThreeDCheckBoxes property which is set to False by default. Setting it to True makes the CheckedListBox to be displayed in Flat or Normal style.

Notable property in the behavior section is the CheckOnClick property which is set to False by default. When set to False it means that to check or uncheck an item in the CheckedListBox we need to double-click the item. Setting it to True makes an item in the CheckedListBox to be checked or unchecked with a single click.

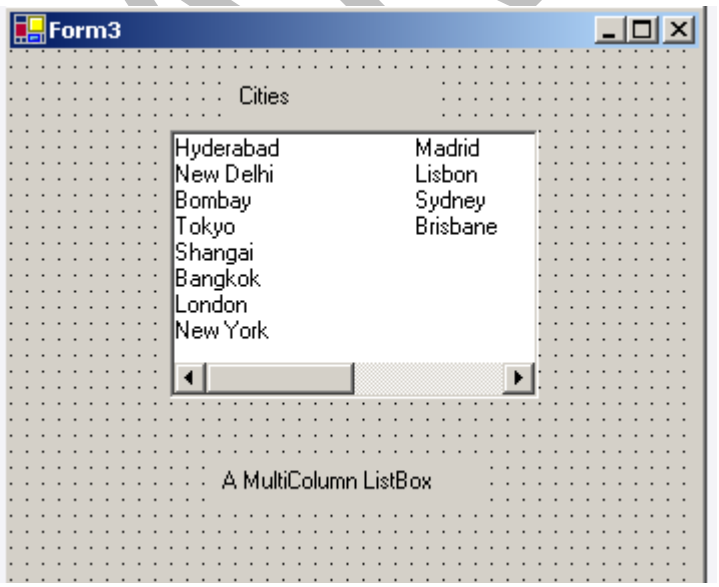
Notable property in the Data section is the Items property with which we add items to the CheckedListBox.

```
Private Sub CheckedListBox1_SelectedIndexChanged(ByVal sender As
System.Object,_
ByVal e As System.EventArgs) Handles
CheckedListBox1.SelectedIndexChanged

End Sub
```

Working with CheckedListBoxes is similar to working with ListBoxes. Please refer to examples in that section here.

ListBox

The ListBox control displays a list of items from which we can make a selection. We can select one or more than one of the items from the list. The ListBox control is based on the ListControl class which is based on the Control class. The image below displays a ListBox.



Notable Properties of the ListBox

In the Behavior Section

HorizontalScrollbar: Displays a horizontal scrollbar to the ListBox. Works when the ListBox has MultipleColumns.
MultiColumn: The default value is set to False. Set it to True if you want the list box to display multiple columns.
ScrollAlwaysVisible: Default value is set to False. Setting it to True will display both Vertical and Horizontal scrollbar always.
SelectionMode: Default value is set to one. Select option None if you do not any item to be selected. Select it to MultiSimple if you want multiple items to be selected. Setting it to MultiExtended allows you to select multiple items with the help of Shift, Control and arrow keys on the keyboard.
Sorted: Default value is set to False. Set it to True if you want the items displayed in the ListBox to be sorted by alphabetical order.

In the Data Section

Notable property in the Data section of the Properties window is the Items property. The Items property allows us to add the items we want to be displayed in the list box. Doing so is simple, click on the ellipses to open the String Collection Editor window and start entering what you want to be displayed in the ListBox. After entering the items click OK and doing that adds all the items to the ListBox.

ListBox Event

The default event of ListBox is the SelectedIndexChanged which looks like this in code:

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As
System.Object, _
ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged

End Sub
```

Working with ListBoxes

Drag a TextBox and a ListBox control to the form and add some items to the ListBox with it's items property.

Referring to Items in the ListBox

Items in a ListBox are referred by index. When items are added to the ListBox they are assigned an index. The first item in the ListBox always has an index of 0 the next 1 and so on.

Code to display the index of an item

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As
```

```
System.Object, _
ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged
TextBox1.Text = ListBox1.SelectedIndex
'using the selected index property of the list box to select the index
End Sub
```

When you run the code and select an item from the ListBox, it's index is displayed in the textbox.

Counting the number of Items in a ListBox

Add a Button to the form and place the following code in it's click event.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles Button1.Click
TextBox1.Text = ListBox1.Items.Count
'counting the number of items in the ListBox with the Items.Count
End Sub
```

When you run the code and click the Button it will display the number of items available in the ListBox.

Code to display the item selected from ListBox in a TextBox

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As
System.Object,_
ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged
TextBox1.Text = ListBox1.SelectedItem
'using the selected item property
End Sub
```

When you run the code and click an item in the ListBox that item will be displayed in the TextBox.

Code to Remove items from a ListBox

You can remove all items or one particular item from the list box.

Code to remove a particular item

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles Button1.Click
ListBox1.Items.RemoveAt(4)
'removing an item by specifying it's index
End Sub
```

Code to Remove all items

```
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
ListBox1.Items.Clear()
'using the clear method to clear the list box
End Sub
```

Panel, GroupBox, PictureBox

Panel

Panels are those controls which contain other controls, for example, a set of radio buttons, checkboxes, etc. Panels are similar to Groupboxes but the difference, Panels cannot display captions where as GroupBoxes can and Panels can have scrollbars where as GroupBoxes can't. If the Panel's Enabled property is set to False then the controls which the Panel contains are also disabled. Panels are based on the ScrollableControl class.

Notable property of the Panel control in the appearance section is the BorderStyle property. The default value of the BorderStyle property is set to None. You can select from the predefined list to change a Panels BorderStyle.
Notable property in the layout section is the AutoScroll property. Default value is set to False. Set it to True if you want a scrollbar with the Panel.

Adding Controls to a Panel

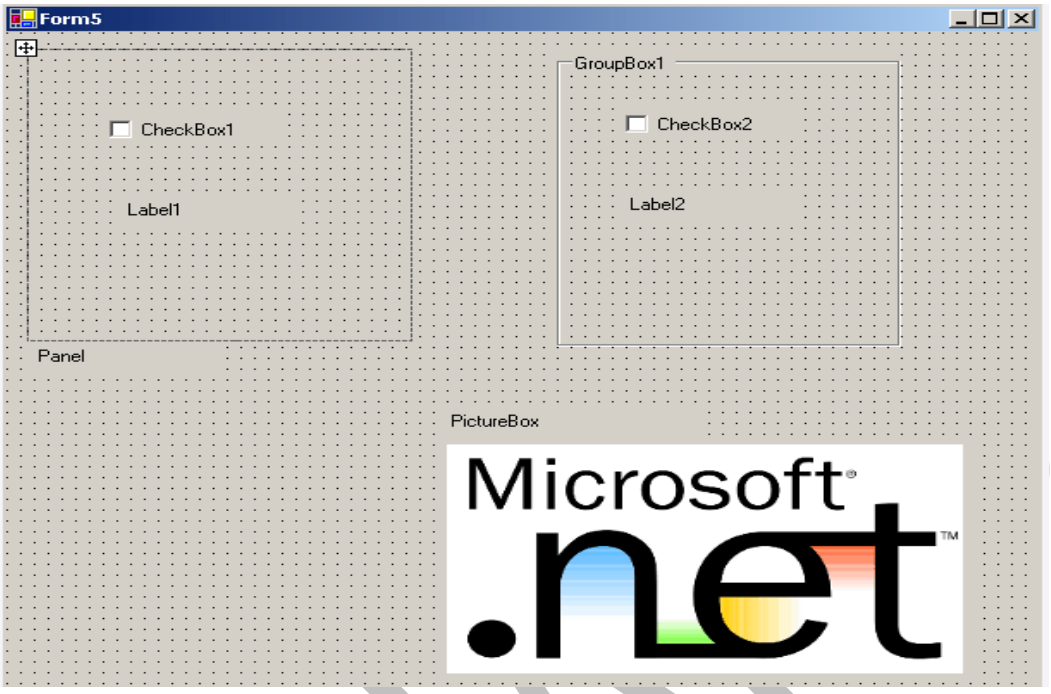
On a from drag a Panel (Panel1) from the toolbox. We want to place some controls, say, checkboxes on this Panel. Drag three checkboxes from the toolbox and place them on the Panel. When that is done all the checkboxes in the Panel are together as in a group but they can function independently.

Creating a Panel and adding a Label and a CheckBox to it in Code

```
Private Sub Form3_Load(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles MyBase.Load
Dim Panel1 As New Panel()
Dim CheckBox1 As New CheckBox()
Dim Label1 As New Label()
Panel1.Location = New Point(30, 60)
Panel1.Size = New Size(200, 264)
Panel1.BorderStyle = BorderStyle.Fixed3D
'setting the borderstyle of the panel
Me.Controls.Add(Panel1)
CheckBox1.Size = New Size(95, 45)
CheckBox1.Location = New Point(20, 30)
CheckBox1.Text = "Checkbox1"
Label1.Size = New Size(100, 50)
Label1.Location = New Point(20, 40)
Label1.Text = "CheckMe"
Panel1.Controls.Add(CheckBox1)
Panel1.Controls.Add(Label1)
```

```
'adding the label and checkbox to the panel
End Sub
```

The image below displays a panel.



GroupBox Control

As said above, Groupboxes are used to Group controls. GroupBoxes display a frame around them and also allows to display captions to them which is not possible with the Panel control. The GroupBox class is based on the Control class.

Creating a GroupBox and adding a Label and a CheckBox to it in Code

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles MyBase.Load
Dim GroupBox1 As New GroupBox()
Dim CheckBox1 As New CheckBox()
Dim Label1 As New Label()
GroupBox1.Location = New Point(30, 60)
GroupBox1.Size = New Size(200, 264)
GroupBox1.Text = "InGroupBox"
'setting the caption to the groupbox
Me.Controls.Add(GroupBox1)
CheckBox1.Size = New Size(95, 45)
CheckBox1.Location = New Point(20, 30)
CheckBox1.Text = "Checkbox1"
label1.Size = New Size(100, 50)
Label1.Location = New Point(20, 40)
Label1.Text = "CheckMe"
GroupBox1.Controls.Add(CheckBox1)
```

```
GroupBox1.Controls.Add(Label1)
'adding the label and checkbox to the groupbox
End Sub
```

PictureBox Control

PictureBoxes are used to display images on them. The images displayed can be anything varying from Bitmap, JPEG, GIF, PNG or any other image format files. The PictureBox control is based on the Control class.

Notable property of the PictureBox Control in the Appearance section of the properties window is the Image property which allows to add the image to be displayed on the PictureBox.

Adding Images to PictureBox

Images can be added to the PictureBox with the Image property from the Properties window or by following lines of code.

```
Private Sub Button1_Click(ByVal sender As System.Object,_
ByVal e As System.EventArgs) Handles Button1.Click
PictureBox1.Image = Image.FromFile("C:\sample.gif")
'loading the image into the picturebox using the FromFile method of
the image class
'assuming a GIF image named sample in C: drive
End Sub
```

ToolTip, ErrorProvider

ToolTip

ToolTips are those small windows which display some text when the mouse is over a control giving a hint about what should be done with that control. ToolTip is not a control but a component which means that when we drag a ToolTip from the toolbox onto a form it will be displayed on the component tray. Tooltip is an Extender provider component which means that when you place an instance of a ToolTipProvider on a form, every control on that form receives a new property. This property can be viewed and set in the properties window where it appears as Tooltip on *n*, where *n* is the name of the ToolTipProvider.

To assign ToolTip's with controls we use it's SetToolTip method.

Notable property of the ToolTip is the Active property which is set to True by default and which allows the tool tip to be displayed.

Setting a ToolTip

Assume that we have a TextBox on the form and we want to display some text when your mouse is over the TextBox. Say the text that should appear is "Do not leave this blank". The code for that looks like this:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles MyBase.Load
ToolTip1.SetToolTip(TextBox1, "Do not leave this blank")
End Sub
```

The image below displays output from above code.

ErrorProvider Component

The ErrorProvider component provides an easy way to set validation errors. It allows us to set an error message for any control on the form when the input is not valid. When an error message is set, an icon indicating the error will appear next to the control and the error message is displayed as Tool Tip when the mouse is over the control.

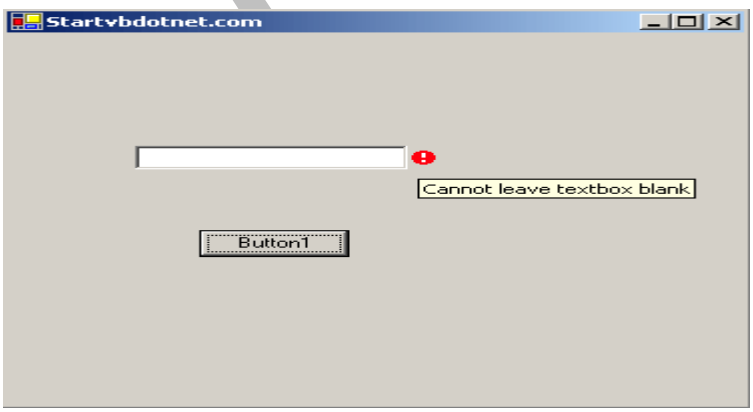
Notable property of ErrorProvider in the Appearance section is the Icon property which allows us to set an icon that should be displayed. Notable property in Behavior section is the BlinkRate property which allows to set the rate in milliseconds at which the icon blinks.

Displaying an Error

Let's work with an example. Assume we have a TextBox and a Button on a form. If the TextBox on the form is left blank and if the Button is clicked, an icon will be displayed next to the TextBox and the specified text will appear in the Tool Tip box when the mouse is over the control. The code for that looks like this:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
If TextBox1.Text = "" Then
ErrorProvider1.SetError(TextBox1, "Cannot leave textbox blank")
Else
ErrorProvider1.SetError(TextBox1, "")
End If
End Sub
```

The image below displays output from above code.



Menus

Everyone should be familiar with Menus. Menus (File, Edit, Format etc in all windows applications) are those that allow us to make a selection when we want to perform some action with the application, for example, to format the text, open a new file, print and so on. In VB .NET MainMenu is the container for the Menu structure of the form. Menus are made of MenuItem objects that represent individual parts of a menu (like File->New, Open, Save, Save As etc). The two main classes involved in menu handling are, MainMenu and MenuItem. The MainMenu class let's us assign objects to a form's menu class and MenuItem is the class which supports the items in a menu system. Menus like File, Edit, Format etc and the items in those Menus are supported by this MenuItem class. It's this MenuItem's click event that makes these Menus work. For a MenuItem to be displayed, we need to add it to a MainMenu object.

Event of the MenuItem

The default event of the MenuItem is the Click event which looks like this in code:

```
Private Sub MenuItem1_Click(ByVal sender As System.Object, ByVal e
As_
System.EventArgs) Handles MenuItem1.Click

End Sub
```

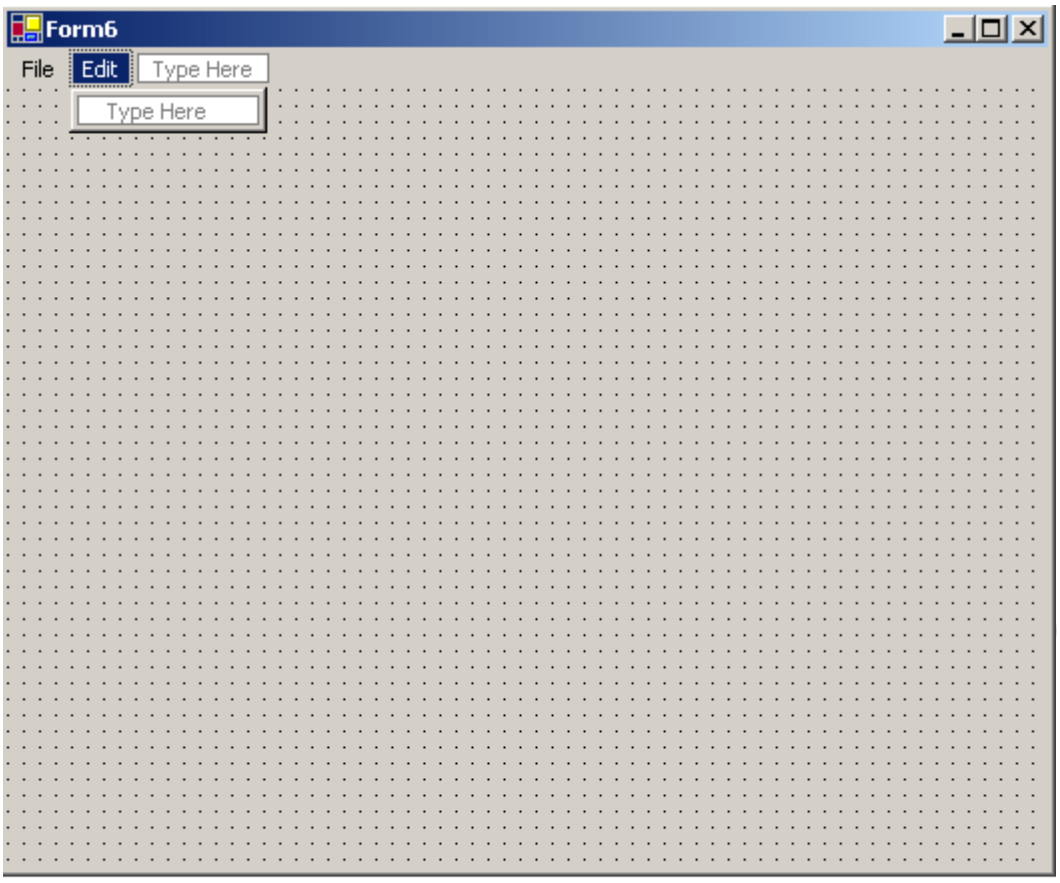
Notable properties of the MenuItem class are summarized below.

Under the Miscellaneous Section of the properties window:

Checked: Default value is set to False. Changing it to True makes a checkmark appear towards the left of the Menu.
DefaultItem: Default value is set to False. Changing it to True makes this menu item default menu item.
RadioCheck: Changing it to True makes a menu item display a radio button instead of a checkmark.
Shortcut: Enables to set a short cut key from a list of available shortcuts for the menu item.

Working with Menus

Creating Menus is simple. Drag a MainMenu component from the toolbar onto the form. When you add a MaiuMenu component to the form it appears in the component tray below the form. Windows form designer will add the MenuItem's for this by default, you need not add this. Once when you finish adding a MainMenu component to the form you will notice a "TypeHere" box towards the top-left corner of the form. To create a menu all you have to do is click on the "TypeHere" text which opens up a small textbox allowing you to enter text for the menu. You can view that in the image below. You can use the arrow keys on the keyboard to create a submenu or add other items to that menu or click on the first menu item and use the left/right arrow keys on the keyboard to create a new menu item. That's all it takes to add a menu to the form.



Working with an example

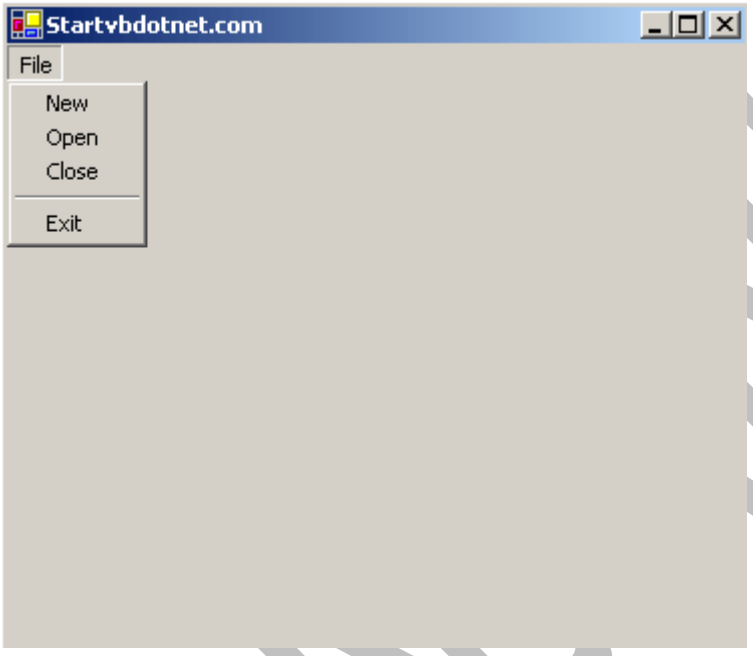
Let's work with an example to understand Menus. Drag a MainMenu and a TextBox onto the form. In the "Type Here" part, type File and under file type "New" and "Exit". Our intention here is to display "Welcome to Menus" in the TextBox when "New" is clicked and close the form when "Exit" is clicked. The Menu which we will create should look like this File->New, Exit (New and Exit below File). The code for that looks like this:

```
Public Class Form3 Inherits System.Windows.Forms.Form
#Region " Windows Form Designer generated code "

Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs)_ Handles MenuItem2.Click
TextBox1.Text = "Welcome to Menus"
End Sub

Private Sub MenuItem3_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs)_ Handles MenuItem3.Click
Me.Close()
'Me refers to the current object (form)
```

```
End Sub

End Class
```

Seperating Menu Items

We can separate menu items with a seperator. A separator is a horizontal line between items on a menu. We can use separator bars to divide menu items into groups on menus that contain multiple items. You add a separator to menus by entering a hypen (-) as the text of a menu item. It will be displayed as a separator. The image below displays a separator bewteen the Close and Exit menu items.

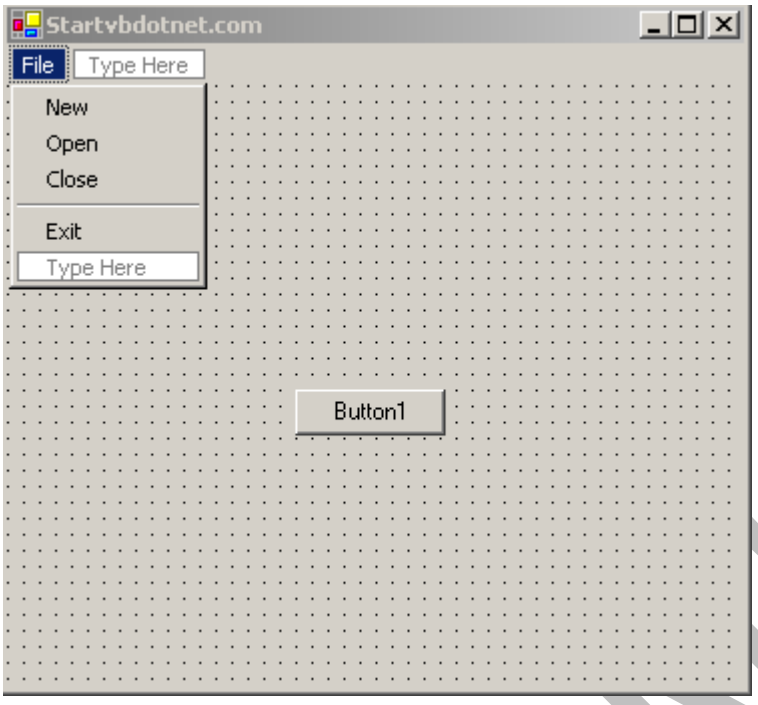


Cloning Menus

Cloning menus is making a copy of existing menu items. For example, we can clone the File menu item displayed in the image below to serve as a context menu for a control. To clone a menu we should use the CloneMenu method. The CloneMenu method creates a copy of the specified menu and all of it's members. This includes their properties and event handlers. Thus, all the events that are handled by the menu item will be handled by the cloned menu. The cloned menu can then be assigned to a control.

Example

On a new form drag a MainMenu component, ContextMenu component and a Button. Click on the MainMenu component and under the "type here" boxes enter the menu items as shown in the image below. The menu items you entered for MainMenu will appear as a context menu (right-click) for the button control. For the purpose of explanation minimum functionality is added to the menu items. The image below displays the form in design view.

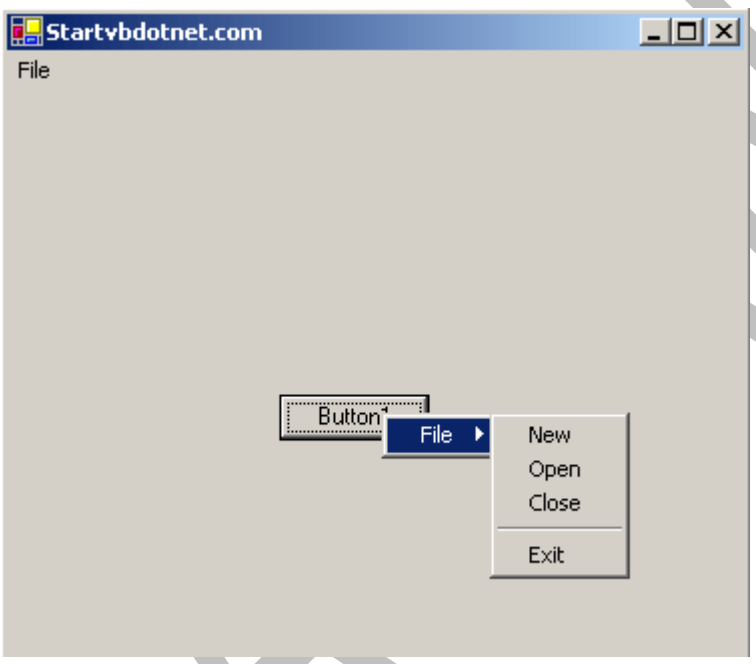


Code to clone the MainMenu component

```
Public Class Form4 Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

#End Region

Private Sub Form4_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
Contextmenu1.MenuItems.Add(MenuItem1.CloneMenu)
'cloning the filemenuitem and filling contextmenu1 with the cloned item
Button1.ContextMenu = Contextmenu1
'assigning the new contextmenu to button1
End Sub

Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles MenuItem2.Click
MessageBox.Show("You clicked New")
End Sub

Private Sub MenuItem3_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles MenuItem3.Click
MessageBox.Show("You clicked Open")
End Sub

Private Sub MenuItem4_Click(ByVal sender As System.Object, ByVal e
```

```
As _
System.EventArgs) Handles MenuItem4.Click
MessageBox.Show("You clicked Close")
End Sub

Private Sub MenuItem6_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles MenuItem6.Click
Me.Close()
End Sub

End Class
```

The image below displays output from the code above. When you right-click the button and click on any menu item, it performs the same functionality as it would for the File menu item.



Shortcut Keys

You can assign shortcut keys to enable instant access to menu commands. To assign a shortcut to the menu item, in the properties window select the shortcut property and choose the appropriate shortcut key combination from the drop-down menu.

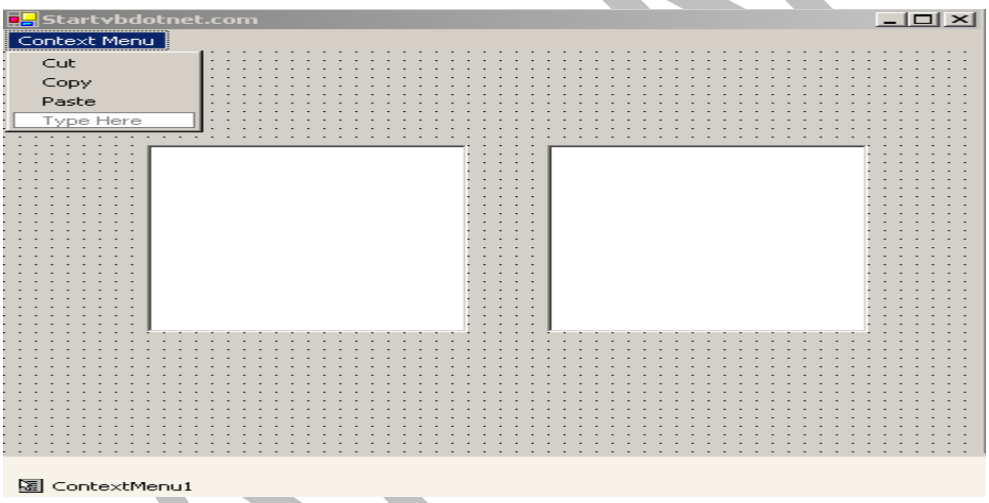
Context Menus

Context menus are menus that appear when an item is right-clicked. In any windows application when you right-click your mouse you get a menu which might display some shortcuts from the Edit Menu, for example, cut, copy, paste, paste special and so on. All these menu items which are

available when you right-click are called Context Menus. In Visual Basic we create context menus with the ContextMenu component. The ContextMenu component is edited exactly the same way the MainMenu component is edited. The ContextMenu appears at the top of the form and you can add menu items by typing them. Toassociate a ContextMenu with a particular form or control we need to set the ContextMenu property of that form or control to the appropriate menu.

Working With Example

Let's understand ContextMenus with an example. On a new Form drag a ContextMenu component from the toolbox. Click on the ContextMenu component to open the editor at the top of the form. In the type here box, enter cut, copy, paste. Cut is assigned MenuItem1, Copy with MenuItem2 and Paste with MenuItem3. Darg two RichTextBoxes onto the form. In the properties window for the form and the richtextboxes, select the ContextMenu property and set it to ContextMenu1. Make sure you set the ContextMenu property for both the richtextboxes. This sample application allows you to enter some text in RichTextBox1, select some text, cut/copy the slected text and paste it in RichTextBox2. It is similar to the right-click menu with which you work in other windows applications. The whole design should look like the image below.

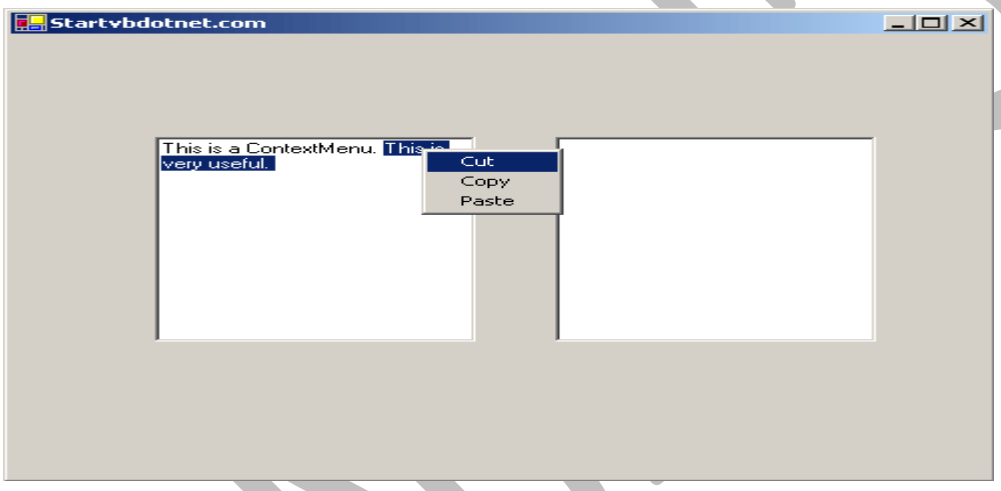


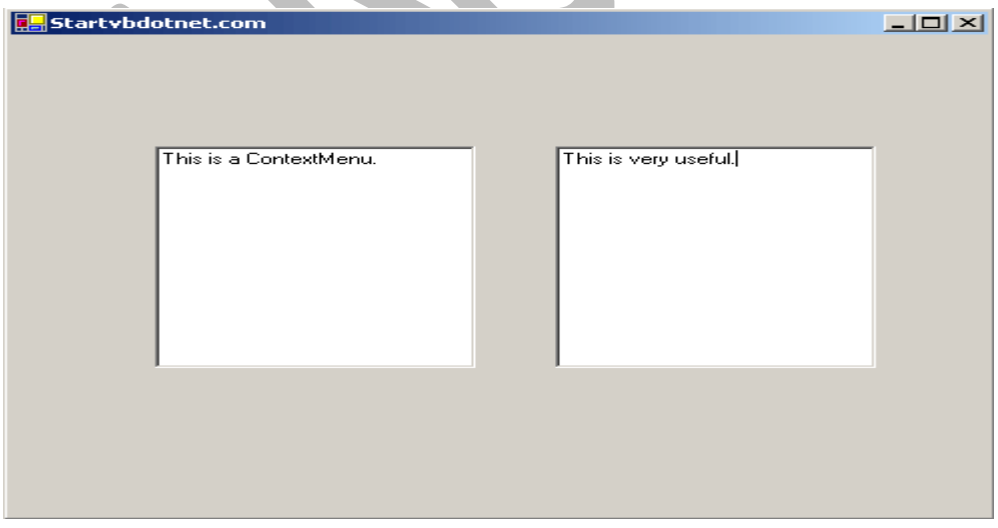
Code to get the desired result looks like this:

```
Public Class Form3 Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

#End Region

Private Sub MenuItem1_Click(ByVal sender As System.Object, ByVal e
_
As System.EventArgs) Handles MenuItem1.Click
RichTextBox1.Cut()
End Sub

Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e
```

```
_
As System.EventArgs) Handles MenuItem2.Click
RichTextBox1.Copy()
End Sub

Private Sub MenuItem3_Click(ByVal sender As System.Object, ByVal e
_
As System.EventArgs) Handles MenuItem3.Click
RichTextBox2.Paste()
End Sub

End Class
```

You can run the application, enter some text in richtextbox1, cut/copy it and paste it in richtextbox2. The images below display sample output from the above said code.





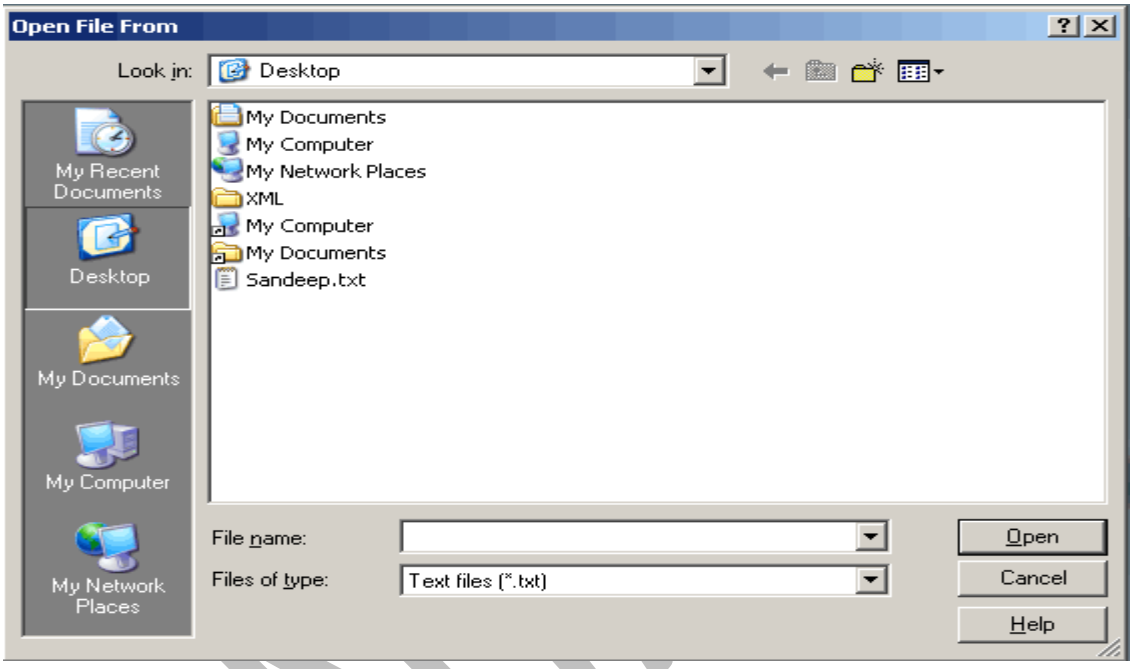
Common Dialogs

Visual Basic .NET comes with built-in dialog boxes which allow us to create our own File Open, File Save, Font, Color dialogs much like what we see in all other windowsapplications. To make a dialog box visible at run time we use the dialog box's ShowDialog method. The

Dialog Boxes which come with Visual Basic .NET are: OpenFileDialog, SaveFileDialog, FontDialog, ColorDialog, PrintDialog, PrintPreviewDialog and PageSetupDialog. We will be working with OpenFile, SaveFile, Font and Color Dialog's in this section. The return values of all the above said dialog boxes which will determine which selection a user makes are: Abort, Cancel, Ignore, No, None, OK, Return, Retry and Yes.

OpenFileDialog

Open File Dialog's are supported by the OpenFileDialog class and they allow us to select a file to be opened. Below is the image of an OpenFileDialog.



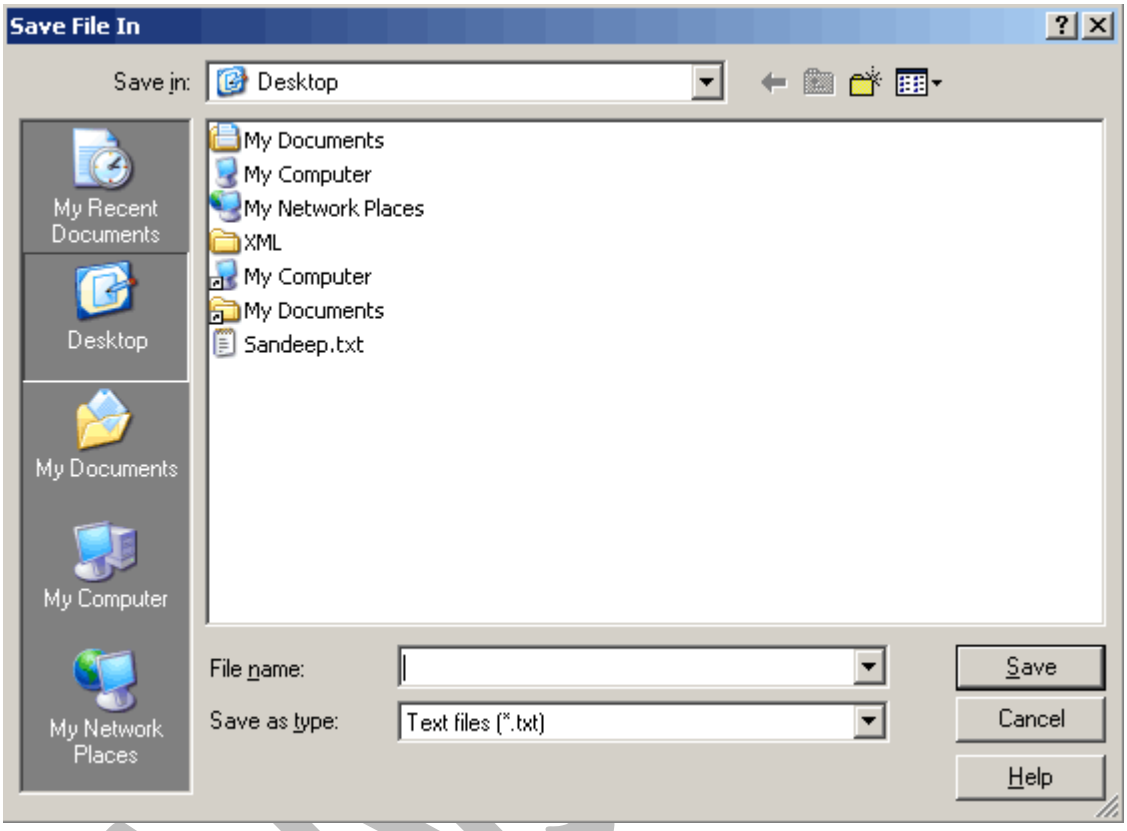
Properties of the OpenFileDialog are as follows:

AddExtension: Gets/Sets if the dialog box adds extension to file names if the user doesn't supply the extension.
CheckFileEixsts: Checks whether the specified file exists before returning from the dialog.
CheckPathExists: Checks whether the specified path exists before returning from the dialog.
DefaultExt: Allows you to set the default file extension.
FileName: Gets/Sets file name selected in the file dialog box.
FileNames: Gets the file names of all selected files.
Filter: Gets/Sets the current file name filter string, which sets the choices that appear in the "Files of Type" box.
FilterIndex: Gets/Sets the index of the filter selected in the file dialog box.
InitialDirectory: This property allows to set the initial directory which should open when you use the OpenFileDialog.
MultiSelect: This property when set to True allows to select multiple file extensions.
ReadOnlyChecked: Gets/Sets whether the read-only checkbox is checked.
RestoreDirectory: If True, this property restores the original directory before closing.
ShowHelp: Gets/Sets whether the help button should be displayed.
ShowReadOnly: Gets/Sets whether the dialog displays a read-only check box.
Title: This property allows to set a title for the file dialog box.

ValidateNames: This property is used to specify whether the dialog box accepts only valid file names.

SaveFileDialog

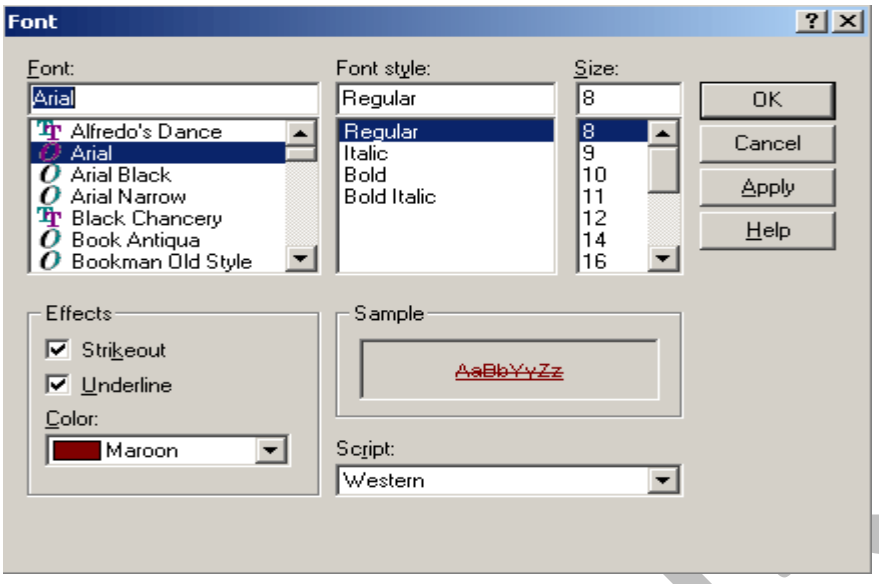
Save File Dialog's are supported by the SaveFileDialog class and they allow us to save the file in a specified location. Below is the image of a SaveFileDialog.



Properties of the Save File Dialog are the same as that of the Open File Dialog. Please refer above. Notable property of Save File dialog is the OverwritePromoptproperty which displays a warning if we choose to save to a name that already exists.

FontDialog

Font Dialog's are supported by the FontDialog Class and they allow us to select a font size, face, style, etc. Below is the image of a FontDialog.



Properties of the FontDialog are as follows:

AllowSimulations: Gets/Sets whether the dialog box allows graphics device interface font simulations.
AllowVectorFonts: Gets/Sets whether the dialog box allows vector fonts.
AllowVerticalFonts: Gets/Sets whether the dialog box displays both vertical and horizontal fonts or only horizontal fonts.
Color: Gets/Sets selected font color.
FixedPitchOnly: Gets/Sets whether the dialog box allows only the selection of fixed-pitch fonts.
Font: Gets/Sets the selected font.
FontMustExist: Gets/Sets whether the dialog box specifies an error condition if the user attempts to select a font or size that doesn't exist.
MaxSize: Gets/Sets the maximum point size the user can select.
MinSize: Gets/Sets the mainimum point size the user can select.
ShowApply: Gets/Sets whether the dialog box contains an apply button.
ShowColors: Gets/Sets whether the dialog box displays the color choice.
ShowEffects: Gets/Sets whether the dialog box contains controls that allow the user to specify to specify strikethrough, underline and text color options.
ShowHelp: Gets/Sets whether the dialog box displays a help button.

ColorDialogs

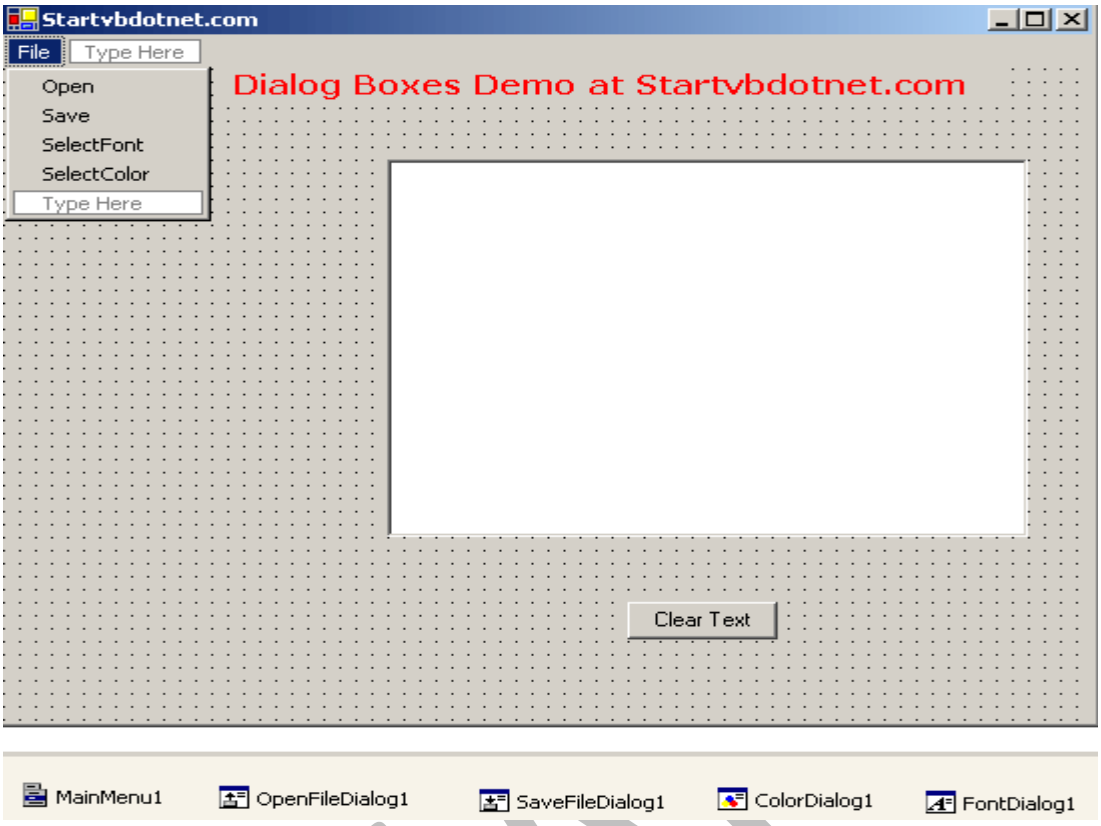
Color Dialog's are supported by the ColorDialog Class and they allow us to select a color. The image below displays a color dialog.

Properties of ColorDialog are as follows:

AllowFullOpen: Gets/Sets whether the user can use the dialog box to define custom colors.
AnyColor: Gets/Sets whether thedialog box displays all the available colors in the set of basic colons.
Color: Gets/Sets the color selected by the user.
CustomColors: Gets/Sets the set of custom colors shown in the dialog box.
FullOpen: Gets/Sets whether the controls used to create custom colors are visible when the dialog box is opened.
ShowHelp: Gets/Sets whether the dialog box displays a help button.
SolidColorOnly: Gets/Sets whether the dialog box will restrict users to selecting solid colors only.

Putting Dialog Boxes to Work

We will work with OpenFile, SaveFile, Font and Color Dialog's in this section. From the toolbox drag a MainMenu component, RichTextBox control, Button Control, OpenFileDialog, SaveFileDialog, FontDialog and ColorDialog onto the form. The sample code demonstrated below allows you to select a file to be opened and displays it in the RichTextBox with OpenFileDialog, allows you to save the text you enter in the RichTextBox to a location using the SaveFileDialog, allows you to select a font and applies the selected font to text in the RTB using FontDialog and allows you to select a color and applies the color to text in the RTB using the ColorDialog. Select the MainMenu component and in the "Type Here" part of the MainMenu type File and using the down arrow keys on the keyboard start typing Open, Save, SelectFont and SelectColor under the File menu. It should look like this: File-> Open, Save, SelectFont, SelectColor. We will assign OpenFileDialog to Open, SaveFileDialog to Save, FontDialog to SelectFont and ColorDialog to SelectColor under File Menu. The form in design view should look similar to the image below.



Before proceeding further you need to set properties for these dialogs in their properties window. They are listed below.

For OpenFileDialog1, set the DefaultExt property to txt so that it opens text files, InitialDirectory property to C:, RestoreDirectory propery to True and the Text property to Open File From.

For SaveFileDialog1, set the DefaultExt property to txt so that it saves files in text format, InitialDirectory property to C: so that when you save a file, it first provides C: drive as the choice of location, OverwritePrompt property to False, RestoreDirectory propery to True and the Text property to Save File In.

For FontDialog1, set the AllowSimulations, AllowVectorFonts, AllowverticalFonts properties to false, MaxSize to 50, MinSize to 5 and ShowApply and ShowColor properties to True.

For ColorDialog1, set AnyColor and SolidColorOnly properties to True.

Code

```
Imports System.IO
Public Class Form1 Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "
```

```
#End Region

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As_
System.EventArgs) Handles Button1.Click
RichTextBox1.Text = " "
'clears the text in richtextbox
End Sub

Private FileName As String
'declaring filename that will be selected
Dim sr As StreamReader
'streamreader is used to read text

Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e
As_
System.EventArgs) Handles MenuItem2.Click
Try
With OpenFileDialog1
'With statement is used to execute statements using a particular object,
here,_
'OpenFileDialog1
.Filter = "Text files (*.txt)|*.txt|" & "All files|*.*"
'setting filters so that Text files and All Files choice appears in the Files of
Type box
'in the dialog
If .ShowDialog() = DialogResult.OK Then
'showDialog method makes the dialog box visible at run time
FileName = .FileName
sr = New StreamReader(.OpenFile)
'using streamreader to read the opened text file
RichTextBox1.Text = sr.ReadToEnd()
'displaying text from streamreader in richtextbox
End If
End With
Catch es As Exception
MessageBox.Show(es.Message)
Finally
If Not (sr Is Nothing) Then
sr.Close()
End If
End Try
End Sub

Private Sub MenuItem3_Click(ByVal sender As System.Object, ByVal e
As_
System.EventArgs) Handles MenuItem3.Click
Dim sw As StreamWriter
'streamwriter is used to write text
Try
With SaveFileDialog1
```

```
.FileName = FileName
.Filter = "Text files (*.txt)|*.txt|" & "All files|*.*"
If .ShowDialog() = DialogResult.OK Then
FileName = .FileName
sw = New StreamWriter(FileName)
'using streamwriter to write text from richtextbox and saving it
sw.Write(RichTextBox1.Text)
End If
End With
Catch es As Exception
MessageBox.Show(es.Message)
Finally
If Not (sw Is Nothing) Then
sw.Close()
End If
End Try
End Sub

Private Sub MenuItem4_Click(ByVal sender As System.Object, ByVal e
As_
System.EventArgs) Handles MenuItem4.Click
Try
With FontDialog1
.Font = RichTextBox1.Font
'initializing the dialog box to match the font used in the richtextbox
.Color = RichTextBox1.ForeColor
'default color is Black
If .ShowDialog = DialogResult.OK Then
setFont()
'calling a method setFont() to set the selected font and color
End If
End With
Catch es As Exception
MessageBox.Show(es.Message)
End Try
End Sub

Private Sub setFont()
Try
With FontDialog1
RichTextBox1.Font = .Font
If .ShowColor Then
RichTextBox1.ForeColor = .Color
'setting the color
End If
End With
Catch ex As Exception
MessageBox.Show(ex.Message)
End Try
End Sub
```

```
Private Sub MenuItem5_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles MenuItem5.Click
Static CustomColors() As Integer = {RGB(255, 0, 0), RGB(0, 255, 0),
RGB(0, 0, 255)}
'initializing CustomColors with an array of integers and putting Red,
Green,
'and Blue in the custom colors section
Try
With ColorDialog1
.Color = RichTextBox1.ForeColor
'initializing the selected color to match the color currently used
'by the richtextbox's foreground color
.CustomColors = CustomColors
'filling custom colors on the dialog box with the array declared above
If .ShowDialog() = DialogResult.OK Then
RichTextBox1.ForeColor = .Color
CustomColors = .CustomColors
'Storing the custom colors to use again
End If
ColorDialog1.Reset()
'resetting all colors in the dialog box
End With
Catch es As Exception
MessageBox.Show(es.Message)
End Try
End Sub

End Class
```

Date TimePicker, Month Calendar, Splitter

Date TimePicker

Date TimePicker allows us to select date and time. Date TimePicker is based on the control class. When we click on the drop-down arrow on this control it displays a monthcalendar from which we can make selections. When we make a selection that selection appears in the textbox part of the Date TimePicker. The image below displays the Date TimePicker.

Notable Properties of Date TimePicker

The Format property in the Appearance section is used to select the format of the date and time selected. Default value is long which displays the date in long format. Other values include short, time and custom

Behavior Section

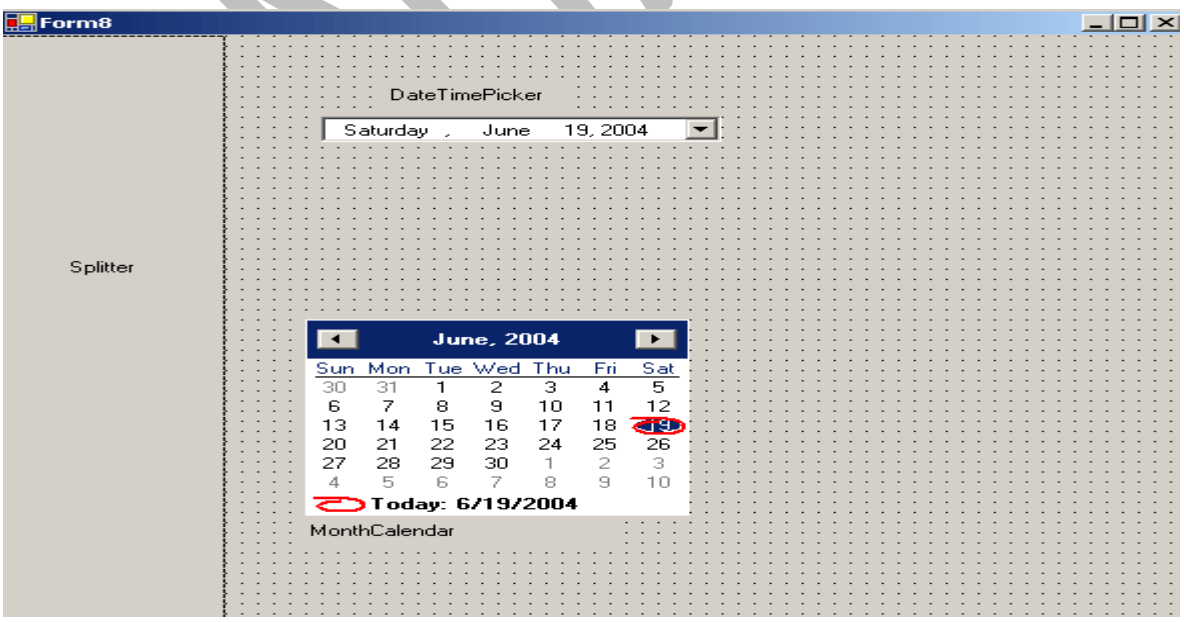
The CustomFormat property allows us to set the format for date and time depending on what we like. To use the CustomFormat property we need to set the Format property to Custom.
The MaxDate Property allows us to set the maximum date we want the Date TimePicker to hold. Default MaxDate value set by the software is 12/31/9998 .
The MinDate Property allows us to set the minimum date we want the Date TimePicker to hold. Default MinDate value set by the software is 1/1/1753 .

MonthCalendar

The MonthCalendar control allows us to select date. The difference between a Date TimePicker and MonthCalendar is, in MonthCalendar we select the date visually and in Date TimePicker when we want to make a selection we click on the drop-down arrow and select the date from the MonthCalendar which is displayed. The image belowdisplays a MonthCalendar control.

Notable Behavior properties of MonthCalendar

FirstDayOfWeek: Default value is Default which means that the week starts with Sunday as the first day and Saturday as last. You can set the first day of the weekdepending upon your choice by selecting from the predefined list with this property.
ShowToday: Default value is set to True which displays the current date at the bottom of the Calendar. Setting it to False will hide it.
ShowTodayCircle: Default value is set to True which displays a red circle on the current date. Setting it to False will make the circle disappear.
ShowWeekNumber: Default is False. Setting it to True will display the week number of the current week in the 52 week year. That will be displayed towards the left side of the control.



Splitter

The Splitter control is used to resize other controls. The main purpose of Splitter control is to save space on the form. Once when we finish working with a particular control we can move it away from it's position or resize them with Splitter control. The Splitter control is invisible when we run the application but when the mouse is over it, the mousecursor changes indicating that it's a Splitter control and it can be resized. This control can be very useful when we are working with controls both at design time and run time (which are not visible at design time). The Splitter control is based on the Control class.

Working with Splitter Control

To work with a Splitter Control we need to make sure that the other control with which this control works is docked towards the same side of the container. Let's do that with an example. Assume that we have a TextBox on the form. Drag a Splitter control onto the form. Set the TextBox's dock property to left. If we want to resize the TextBox once we finish using it set the Splitter's dock property to left (both the controls should be docked towards the same end). When the program is executed and when you passthe mouse over the Splitter control it allows us to resize the TextBox allowing us to move it away from it's current position.

HelpProvider

Providing help with your application is very useful as it allows your users to understand it more easily, thereby increasing productivity and saving some money. Support for help in Visual Basic exists and you can display HTML files that can contain a set of linked topics.

Help Class

The Help class allows us to display HTML help to users. The Help class provides two methods: ShowHelp and ShowHelpIndex. ShowHelp is used to display a help file for aparticular control and requires that control to be displayed along with the help file. The URL you specify for the help file can be in the form of C:\myHelp (local machine) or http:\\www.startvbdotnet.com\help.htm (Web). Generally, the Help class is used in conjunction with Menus, Context Menus.

Example

The following code displays a help file. This code assumes that you have a Button, Button1, Label, Label1 on the form and a help file named Apphelp.htm in the C: drive of your machine.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e_
As System.EventArgs) Handles Button1.Click
Help.ShowHelp(Label1, "C:\Apphelp.htm")
'using the Help class with label control
End Sub
```

The ShowHelpIndex method is used to display the index of a specified help file. You call the ShowHelpIndex method just like you call the ShowHelp method.

HelpProvider Component

HelpProviderComponent allows us to provide help for controls on the form when F1 key is pressed. The HelpProviderComponent is an extender provider which means that it coordinates and maintains properties for each control on the form. Notable Property of HelpProvider component is the HelpNameSpace property which specifies the URL for the help file associated with it.

The HelpProvider component provides three additional properties to each control on the form. These properties are:
HelpString: Determines the help string associated with a control.
HelpKeyWord: Determines the help keyword associated with a control.
HelpNavigator: Determines the kind of help associated with a control. Provides six values: TableOfContents, Find, Index, Topic, AssociatedIndex and KeywordIndex.

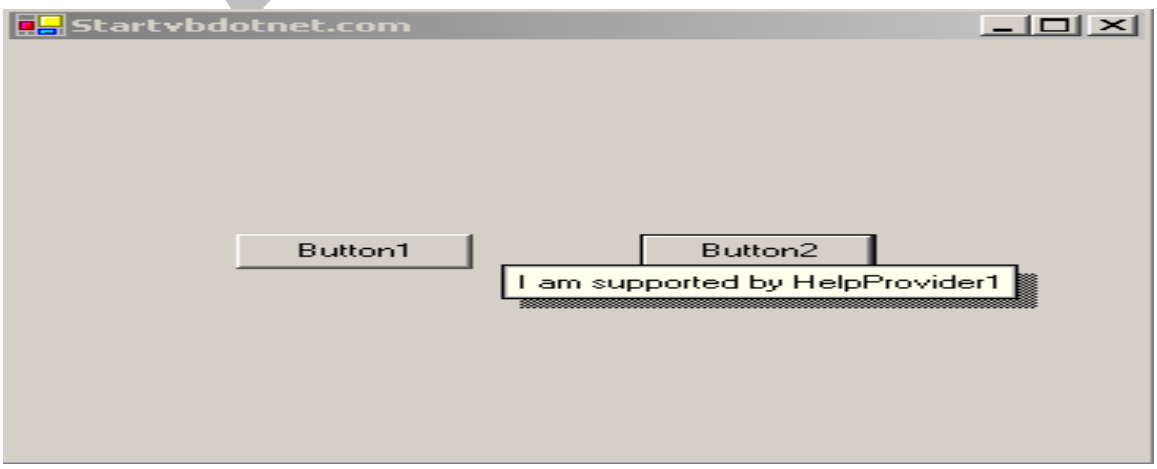
The above said three properties are visible in the properties window for each control once the HelpProvider component is added to the form. If the HelpNameSpace property is not set, the HelpString is automatically displayed, and other two properties are ignored.

Example

Drag two Buttons and the HelpProvider component onto the form. The following code displays a help string when Button2 has the focus and F1 key is pressed.

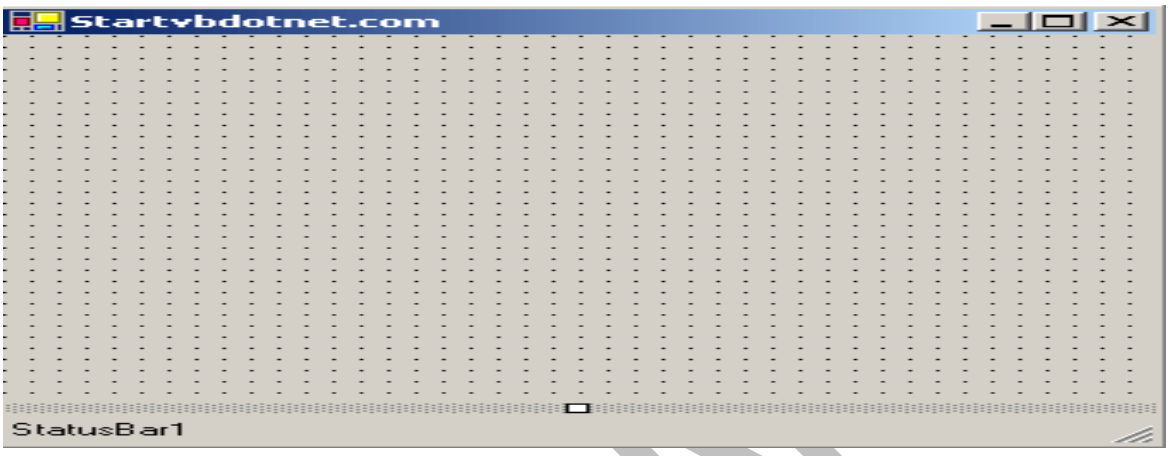
```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
HelpProvider1.SetHelpString(Button2, "I am supported by
HelpProvider1")
'button2 needs to have focus to display this string when F1 is pressed
End Sub
```

The image below displays output from code above.



StatusBar

Status Bars are used to display status messages at the bottom of the form. They are generally used to provide additional information, such as page numbers, display amessage, etc. There are two kinds of status bars: simple status bars and status bars that display a panel. Simple status bars display a single message on the status bar and a status bar with panels can display multiple messages. Below is the image of a StatusBar control.



Notable properties of the Status bar:

Panels: Gets the collection of status bar panels in a status bar.
ShowPanels: Default is True. You can set it to False if you don't want to show panels.
Text: Gets/Sets the text to be displayed.

StatusBar Event

Defalut event of the StatusBar control is the PanelClick event which looks like this in Code:

```
Private Sub StatusBar1_PanelClick(ByVal sender As System.Object,
ByVal e _
As System.Windows.Forms.StatusBarPanelClickEventArgs) Handles
StatusBar1.PanelClick

End Sub
```

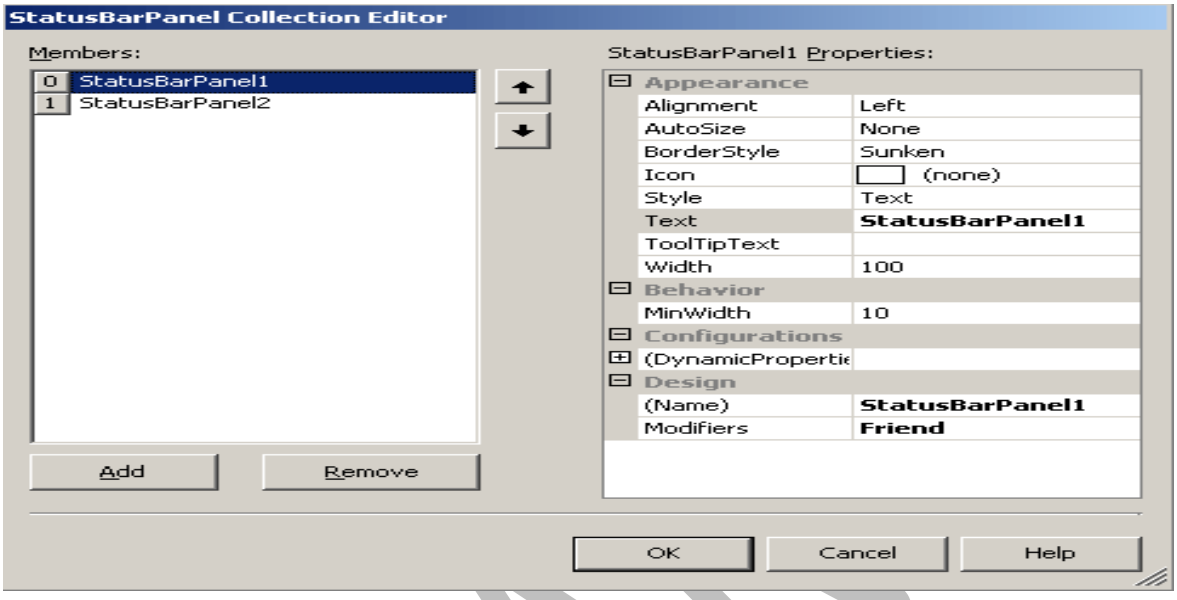
Sample code for a Simple Status Bar

From the toolbox add a status bar control to the form. The following code will display the message "Hello" on the StatusBar when the form loads.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
StatusBar1.Text = "Hello"
End Sub
```

Status Bars with Panels

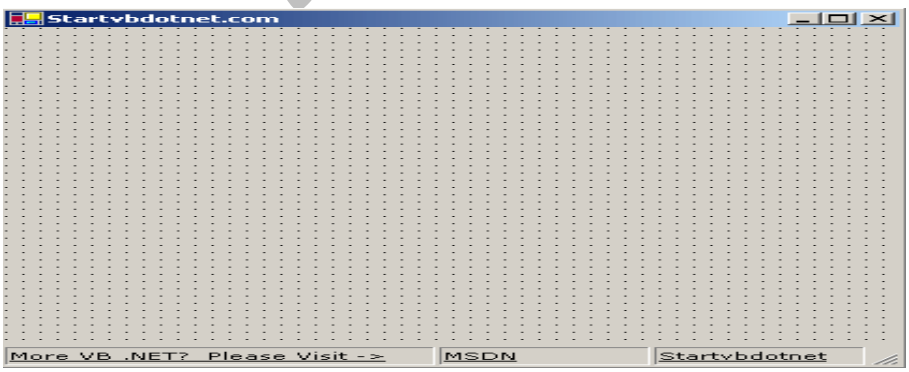
You can add panels to a status bar by opening it's properties window and clicking the Panels property. When you click the Panels property it opens the StatusBarPanel Collection Editor which looks like the image below.



You add panels by clicking the Add button found in the editor. While adding panels you can set the Text to be displayed for each panel, an icon, tooltip, width for each panel you add.

To add panels to status bar in code we use the **StatusBar.Panel.Add** method and **StatusBar.Panels.Remove**, **StatusBar.Panels.RemoveAt** to remove the panels. To access text in each panel you use the text property of StatusbarPanel as: **StatusBarPanels(0).Text="I am panel one".**

To handle status bar panel clicks you use the PanelClick event as shown in the code below. To work with this code, add a status bar control to the form, open it's properties window, select the Panels property and add three status bar panels. For StatusBarPanel1 set the text "More VB .NET? Please Visit ->", for StatusBarPanel2 set the text "MSDN" and for StatusBarPanel3 "Startvbdotnet.com". The form in design view should look like the image below.



Switch to code view and paste the following code:

```
Private Sub StatusBar1_PanelClick(ByVal sender As System.Object,
ByVal e As _
System.Windows.Forms.StatusBarPanelClickEventArgs) Handles
StatusBar1.PanelClick
If e.StatusBarPanel Is StatusBar1.Panels(1) Then
'checks if status bar panel2 is clicked and if true opens a webpage
System.Diagnostics.Process.Start("www.msdn.mcirosoft.com")
ElseIf e.StatusBarPanel Is StatusBar1.Panels(2) Then
'checks if status bar panel3 is clicked and if true opens a webpage
System.Diagnostics.Process.Start("www.startvbdotnet.com")
End If
End Sub
```

When you run the application and click on MSDN you will be taken to MSDN and Startvbdotnet.com if you click on Startvbdotnet.

NotifyIcon

Notify Icons display an icon in Windows System Tray. This is really useful for processes that run in the background and don't have their own interface. Since VB allows us to create Windows Services (services that run in the background and display control panels) now, we can use these notify icon's to associate funtionality to windows services. You can also use this icon to associate help with your application, launch another application or anything else which you think can be appropriate.

Notable properties of Notify Icon:

ContextMenu: Gets/Sets Context menu for the tray icon
Icon: Gets/Sets current icon
Text: Gets/Sets tooltip text that is displayed when the mouse hovers over the system tray
Visible: Gets/Sets if the icon is visible in the windows system tray

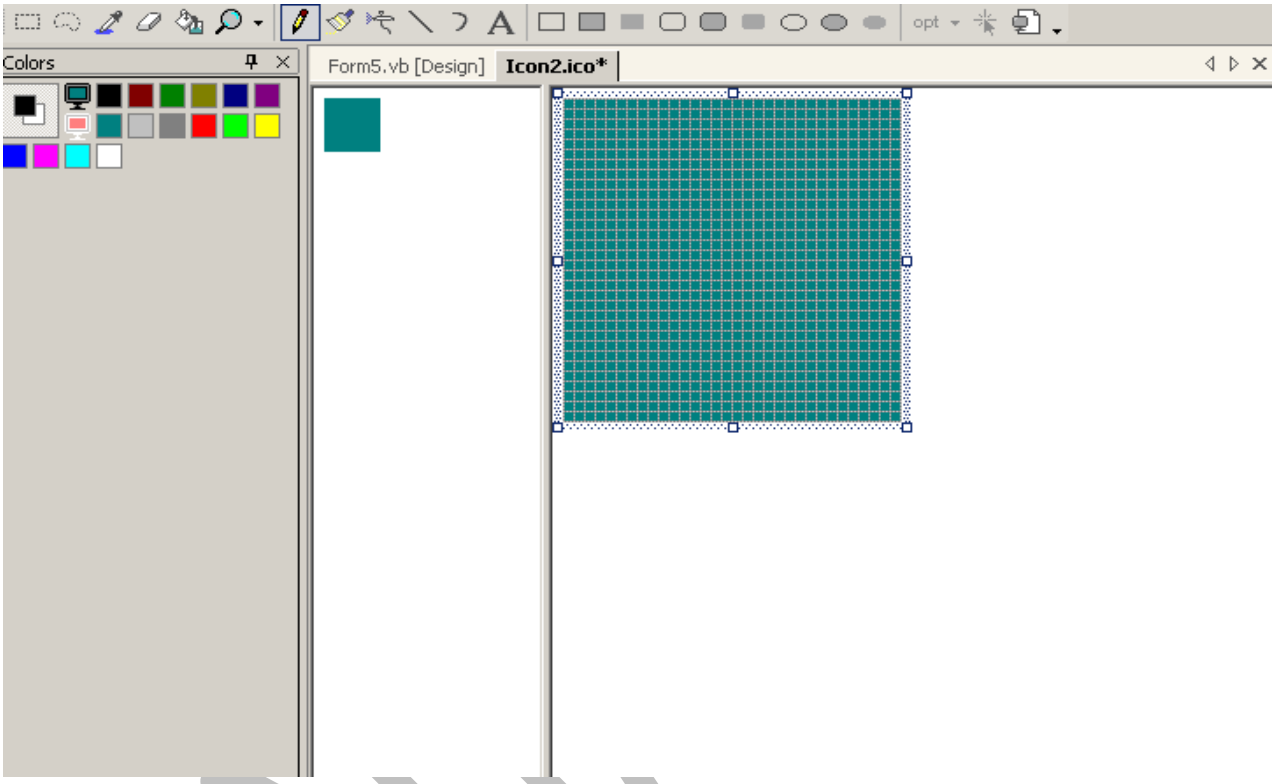
Notify Icon Event

The default event associated with Notify Icon is the MouseDown event which looks like this in code:

```
Private Sub NotifyIcon2_MouseDown(ByVal sender As System.Object,
ByVal e As _
System.Windows.Forms.MouseEventArgs) Handles
NotifyIcon2.MouseDown

End Sub
```

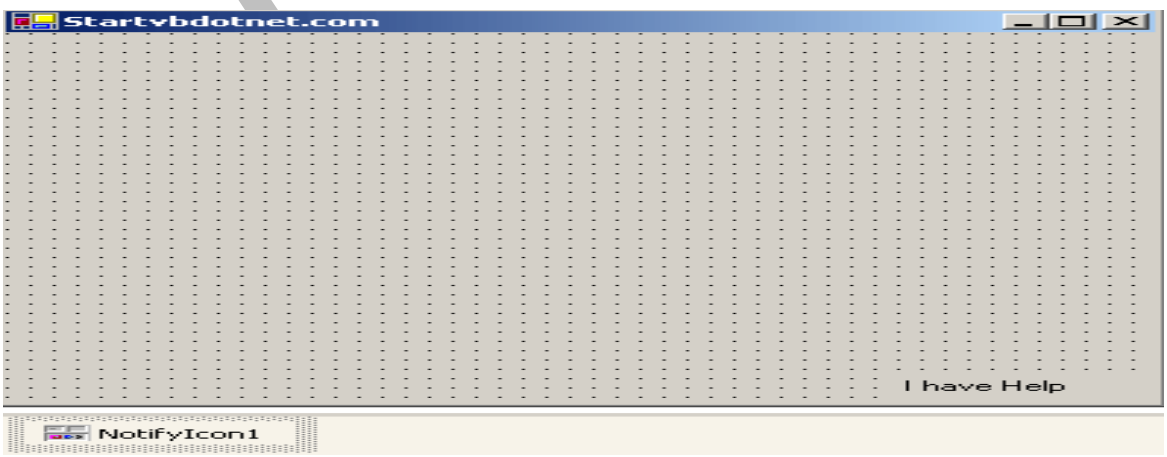
You can also handle click and double-click events for notify icon. The code sample below works with the click event of the Notify Icon to display a help file.

To create a Notify Icon component you need an icon (.ico) file to assign to it's Icon property. If you have an icon then you can use it else you might need to create an icon. You can create new icons with Visual Studio's icon designer. To open the icon designer select Project->Add New Item and from the Add New Item dialog select Icon File and click open. You can use the toolbars that are visible to design your icon. The Icon Designer Window is displayed below.



Sample Code

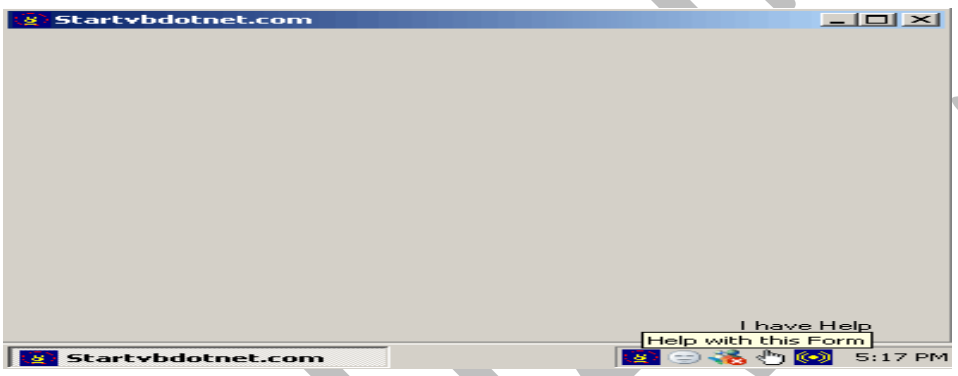
Drag a Notify Icon component and a Label control from the toolbar onto the form. Open the properties window for the Notify Icon and set the Icon property to the path of the icon and the text property to "Help with this Form". This is the icon that will be displayed when you run the application. The Label control is needed to set the help file. Set the text for label as "I have Help". The form in design view should look like the image below.



This sample code launches a help file when you click the Icon in System Tray.
This sample code assumes that you have a help file, "Help.htm" in the C: drive of your machine.

```
Private Sub NotifyIcon1_click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles NotifyIcon1.Click
'handling click event of the NotifyIcon
Help.ShowHelp(Label1, "c:\help.htm")
'using the Help class and it's ShowHelp method to display a help file
End Sub
```

When you run the application, an icon will be visible in the System Tray and when you click the icon the help file named "Help.htm" will be launched. The image below displays the output from above code.



Print Dialogs

Providing support for Printing is one of the common tasks during application development. The .NET Framework provides excellent support for Printing documents.

PrintDocument Component

In the .NET Framework, a printed document is represented by the PrintDocument component. The PrintDocument object encapsulates all the information needed to print a page. They handle the events and operations of printing. The PrintDocument object exposes three properties which are as follows:

PrinterSettings Property: Contains information about the capabilities and settings of the printers.
DefaultPageSettings Property: Encapsulates the configuration for printing each printed page.
PrintController Property: Describes how each page is guided through the printing process.

How Printing Works?

Printing content is provided directly by the application logic in the .NET Framework. You add a PrintDocument object to the project and handle the PrintPage event which is called every time a new page is to be printed. A print job is initiated by the PrintDocument's Print method. This starts the print job and raises one or more events. When

the print job begins, a BeginPrint event occurs, followed by the PrintPage event for each page, followed by the EndPage event when the job is done. If the print job contains multiple pages, one PrintPage event will be raised for each page in the job making the PrintPage event to execute multiple times. The PrintPage event is the main eventinvolved in printing documents. To send content to the printer you must handle this event and provide code to render the content in the PrintPage event handler.

PrintDialogs

Print dialogs are supported by the PrintDialog class and they allow us to print the document. To print a document we need to set the Document property of the PrintDialog to PrintDocument object and PrinterSettings to PrinterSettings object. You can print the document by assigning PrintDialog object's PrinterSettings property to the PrinterSettings property of the PrintDocument object and use the PrintDocument object's Print method.

Properties of the Print Dialog are as follows:

AllowPrintTofile: Gets/Sets whether the Print to file checkbox is enabled.
AllowSelection: Gets/Sets whether the selection radio is enabled.
AllowSomePages: Gets/Sets whether the From...To...Page radio button is enabled.
Document: Gets/Sets the PrintDocument used to obtain PrinterSettings.
PrinterSettings: Gets/Sets the PrinterSettings dialog box to modify.
PrintToFile: Gets/Sets whether the Print to file checkbox is enabled.
ShowHelp: Gets/Sets whether the Help button is displayed.
ShowNetwork: Gets/Sets whether the network button is displayed.

PrintPreviewDialog

Print Preview dialogs are supported by the PrintPreviewDialog class and they allow us to preview the document before printing. You can preview a document by setting theDocument property of the PrintPreviewDialog object to the PrintDocument object. Once set, the PrintPreviewDialog provides functionality to zoom, print, to view multiple pages of the preview, etc.

PrintPreviewControl

PrintPreview control allows to create our own custom previews. They display print previews and you can use it to create your own custom print preview windows. To use this control you need to set the print document to it's Document property.

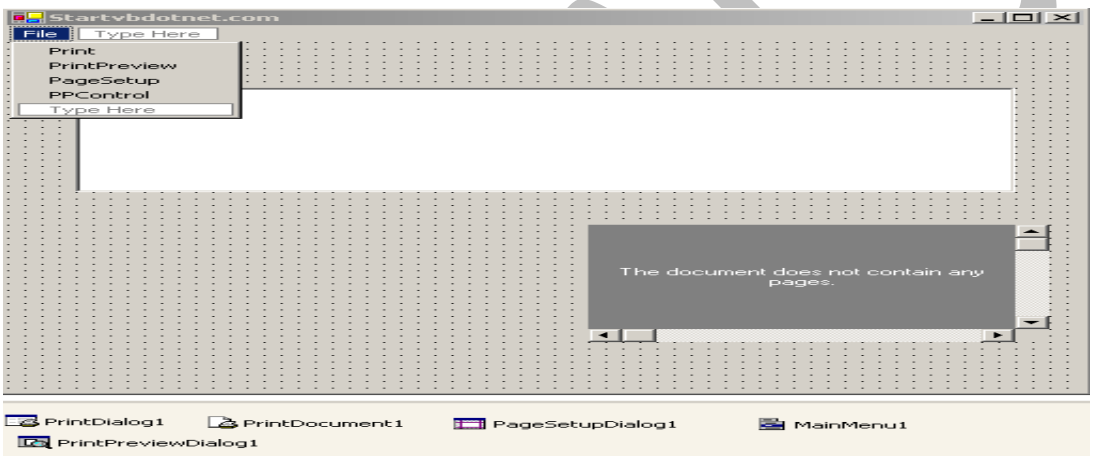
Notable properties of the PrintPreviewControl are as follows:

AutoZoom: When True (default), resizing the control automatically zooms to make all contents visible.
Columns: Gets/Sets the number of pages displayed horizontally.
Rows: Gets/Sets the number of pages displayed vertically.
StartPage: Gets/Sets the apge number of the upper left page.
Zoom: Gets/Sets a value specifying how large the pages will appear.

PageSetupDialog

Page Setup dialogs are used to specify page details for printing. This dialog allows us to set borders and adjustments, headers and footers, portraits, orientation, etc. You use the PrinterSettings property of this dialog to get a Printer Settings object that holds settings the user specified. The PageSetupDialog exposes a Document property that specifies the PrintDocument to use. Setting the Document property binds the specifed PrintDocument to the PageSetupDialog and any changes made in this dialog are updated in the PrinterSettings property.

Let's work with print related controls provided by the .NET Framework. On a new form drag a PrintDialog, PrintDocument, PrintPreviewControl, PrintPreviewDialog, PageSetupDialog, MainMenu and a RichtTextBox contol. Select MainMenu and In the "Type Here" part, type File and under file type Print, PrintPreview, PageSetup and PPControl. The menu should look like this:File->Print, PrintPreview, PageSetup, PPControl. The RichTextBox control is used to load some text in it which will be ready to print. The Form in design view should look like the image below.



Before proceeding further you need to set properties for these dialogs in their properties window. You can set these properties at run time if you wish. Setting them at design time will reduce some lines of code. The necessary changes are listed below.

For PrintDialog1, set the AllowSelection and AllowSomePages properties to True and the Document property to PrintDocument1.

For PrintPreviewDialog1, PageSetupDialog1 and PrintPreviewControl1, set the Document property to PrintDocument1 (for all of them).

Code

```
Imports System.Drawing.Printing
Public Class Form1 Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

#End Region
```

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
RichTextBox1.Text = "Programmers have undergone a major change in
many years of _
programming various machines. For example, what could take days to
create an _
application in other programming languages like C, C++ could be done in
hours with _
Visual Basic. Visual Basic provides many interesting sets of tools to aid us
in _
building exciting applications. Visual Basic provides these tools to make
our _
life far more easier because all the real hard code is already written for
us."
'filling the richtextbox with some text that can be used readily
End Sub

Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles MenuItem2.Click
If PrintDialog1.ShowDialog = DialogResult.OK Then
'showDialog method makes the dialog box visible at run time
PrintDocument1.Print()
End If
End Sub

Private Sub MenuItem3_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles MenuItem3.Click
Try
PrintPreviewDialog1.ShowDialog()
Catch es As Exception
MessageBox.Show(es.Message)
End Try
End Sub

Private Sub MenuItem4_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles MenuItem4.Click
With PageSetupDialog1
.PageSettings = PrintDocument1.DefaultPageSettings
End With
Try
If PageSetupDialog1.ShowDialog = DialogResult.OK Then
PrintDocument1.DefaultPageSettings = PageSetupDialog1.PageSettings
End If
Catch es As Exception
MessageBox.Show(es.Message)
End Try
```

```
End Sub

Private Sub MenuItem5_Click(ByVal sender As System.Object, ByVal e
As _
System.EventArgs) Handles MenuItem5.Click
Try
PrintPreviewControl1.Document = PrintDocument1
Catch es As Exception
MessageBox.Show(es.Message)
End Try
End Sub

Private Sub PrintDocument1_PrintPage(ByVal sender As Object, ByVal e
As_
System.Drawing.Printing.PrintPageEventArgs) Handles
PrintDocument1.PrintPage
'PrintPage is the foundational printing event. This event gets fired for
every
' page that will be printed
Static intCurrentChar As Int32
' declaring a static variable to hold the position of the last printed char
Dim font As New Font("Verdana", 14)
' initializing the font to be used for printing
Dim PrintAreaHeight, PrintAreaWidth, marginLeft, marginTop As Int32
With PrintDocument1.DefaultPageSettings
' initializing local variables that contain the bounds of the printing area
rectangle
PrintAreaHeight = .PaperSize.Height - .Margins.Top - .Margins. Bottom
PrintAreaWidth = .PaperSize.Width - .Margins.Left - .Margins.Right
' initializing local variables to hold margin values that will serve
' as the X and Y coordinates for the upper left corner of the printing
' area rectangle.
marginLeft = .Margins.Left
marginTop = .Margins.Top
' X and Y coordinate
End With

If PrintDocument1.DefaultPageSettings.Landscape Then
Dim intTemp As Int32
intTemp = PrintAreaHeight
PrintAreaHeight = PrintAreaWidth
PrintAreaWidth = intTemp
' if the user selects landscape mode, swap the printing area height and
width
End If

Dim intLineCount As Int32 = CInt(PrintAreaHeight / font.Height)
' calculating the total number of lines in the document based on the height
of
' the printing area and the height of the font
```

```
Dim rectPrintingArea As New RectangleF(marginLeft, marginTop,
PrintAreaWidth, PrintAreaHeight)
' initializing the rectangle structure that defines the printing area
Dim fmt As New StringFormat(StringFormatFlags.LineLimit)
'instantiating the StringFormat class, which encapsulates text layout
information
Dim intLinesFilled, intCharsFitted As Int32
e.Graphics.MeasureString(Mid(RichTextBox1.Text, intCurrentChar + 1),
font,_
New SizeF(PrintAreaWidth, PrintAreaHeight), fmt, intCharsFitted,
intLinesFilled)
' calling MeasureString to determine the number of characters that will fit
in
' the printing area rectangle
e.Graphics.DrawString(Mid(RichTextBox1.Text, intCurrentChar + 1),
font,_
Brushes.Black, rectPrintingArea, fmt)
' print the text to the page
intCurrentChar += intCharsFitted
'advancing the current char to the last char printed on this page
< TextBox1.Text.Length Then
If intCurrentChar e.HasMorePages=True
'HasMorePages tells the printing module whether another PrintPage event
should be fired
Else
e.HasMorePages = False
intCurrentChar = 0
End If
End Sub

End Class
```

The above code will throw exceptions if you don't have a printer attached to your machine.