Modern Programming Tools and Techniques-III

DCAP301

Edited by:
Kumar Vishal

LOVELY PROFESSIONAL UNIVERSITY

LOVELY PROFESSIONAL UNIVERSITY

MODERN PROGRAMMING TOOLS & TECHNIQUES-III

Edited By

Kumar Vishal

Printed by
EXCEL BOOKS PRIVATE LIMITED
A-45, Naraina, Phase-I,
New Delhi-110028
for
Lovely Professional University
Phagwara

SYLLABUS

Modern Programming Tools & Techniques-III

Objectives:

- To impart the skills needed to implement Network enabled technologies.
- To enable the student to understand dot net framework classes.
- To enable the student to develop multi-language support applications.
- To enable the student to develop platform independent applications.
- To enable the student to develop console and windows applications.
- To enable the student to implementing Object oriented concepts in dot net.
- To enable the student to learn file handling using dot net.
- To enable the student to understand Database application development using dot net and data transmission technology.

S.No.	Description
1.	***Introduction:*** What is VB.NET, Characteristics of VB.NET, VB.NET as a language in .NET Framework.
2.	***Variables and Data Types:*** Variables and Data Types. Decision Making and Looping: If, If else if. While, do while, for loop, Declaring Arrays. System. Array class.
3.	***Built-in Functions:*** String Class, Conversion functions, other Miscellaneous Functions, Subroutines and Functions.
4.	***Classes & Object in VB.NET:*** Using Classes, object, methods. Constructors. Creating Properties and indexers. Using Inheritance in classes.
5.	***Namespaces:*** Meaning and its working. Using System Namespace and Object class. Exception Handling: Using Try and Catch blocks, The Finally Section.
6.	***Using System. Collections:*** Array List, Stack, Queue, Sorted List etc.
7.	***Windows Programming:*** Using Controls- textboxes, listbox, buttons, datetime picker, comboboxes etc.
8.	***Common Dialog Boxes:*** OpenFileDialog, SaveFileDialog, ColorDialog, MessageBox Class and DialogResult Class.
9.	***File Input/Output:*** Working with Files and Directories. System.IO.
10.	***ADO.NET:*** Accesing Database with ADO.NET. Executing Insertion, deletion, updation and select command with databases. XML Basics: What is XML? Data Representation through XML. Working with XMLReader and XMLWriter Classes.

CONTENT

Unit 1: Introduction to Visual Basic

Notes

CONTENTS

Contd...

Notes

Objectives

After studying this unit, you will be able to:

- Discuss visual basic elements
- Explain object-oriented programming in VB.NET
- Understand visual basic.net framework
- Discuss common type system
- Elaborate VB.NET features
- Explain .NET framework
- Understand VB.NET as a language in .NET framework

Introduction

Computer programming, often shortened to programming, scripting, or coding is the process of designing, writing, testing, debugging, and maintaining the source code of computer programs. This source code is written in one or more programming languages such as C++, C#, Java, Python, Smalltalk, etc. The purpose of programming is to create a set of instructions that computers use to perform specific operations or to exhibit desired behaviors. The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithms and formal logic. Within software engineering, programming is regarded as one phase in a software development process.

Visual basic is a high level programming language developed from the earlier DOS version called BASIC. Visual Basic .NET is the latest technology introduced by Microsoft with tons of new features including the .NET framework. Educational institutes, Universities and Software Development companies have migrated to VB.NET now but Visual Basic 6 is still widely learned and taught. Learning Visual Basic 6 is quite easier than other programming languages such as C++, C#, Java etc. This is because Visual Basic enables you to work in a graphical user interface where you can just drag and drop controls that you want to work with where you have to write bunches of code to create in C++ or C# or even in Java. If you are new to programming and want to start it in the smoothest and easiest way, then you should start it with Visual Basic.

Sometime in the July 2000, Microsoft announced a whole new software development framework for Windows called .NET in the Professional Developer Conference (PDC). Microsoft also released PDC version of the software for the developers to test. After initial testing and feedback Beta 1 of .NET was announced. Beta 1 of the .NET itself got lot of attention from the developer community. When Microsoft announced Beta 2, it incorporated many changes suggested by the community and internals into the software. The overall 'Beta' phase lasted for more than 1 ½ years. Finally, in March 2002 Microsoft released final version of the .NET framework. One thing to be noted here is the change in approach of Microsoft while releasing this new platform.

Notes

Unlike other software where generally only a handful people are involved in beta testing, .NET was thrown open to community for testing in its every pre-release version. This is one of the reasons why it created so many waves of excitement within the community and industry as well. Microsoft has put in great efforts in this new platform. In fact Microsoft says that its future depends on success of .NET. The development of .NET is such an important event that Microsoft considers it equivalent to transition from DOS to Windows. All the future development – including new and version upgrades of existing products – will revolve around .NET. So, if you want to be at the forefront of Microsoft Technologies, you should be knowing .NET.

1.1 Elements of Visual Basic

The two basic elements of Visual Basic are the GUI and the code associated with the application that makes it respond to events occurred as a result of a user action.

1.1.1 Visual Element

IDE is a term commonly used in the programming world to describe the interface and environment that we use to create our applications. It is called integrated because we can access virtually all of the development tools that we need from one screen called an interface. The IDE is also commonly referred to as the design environment, or the program.

The Visual Basic IDE is made up of a number of components:

- Menu Bar
- Tool Bar
- Project Explorer
- Properties Window
- Form Layout Window
- Toolbox
- Form Designer
- Object Browser

Figure 1.1: VB Startup Window

Source: http://visualbasic.freetutes.com/learn-vb6/lesson1.html

Notes

1.1.2 Language Element

Microsoft Visual Basic code is written in units called procedures. A procedure contains a series of Visual Basic statements that perform an operation or calculate a value. An event procedure is a procedure that runs in response to an event initiated by the user or program code, or triggered by the system. Its syntax is,

```
Private Sub button1_Click()
//code
End Sub
```

Self Assessment

Fill in the blanks:

1. The purpose of programming is to create a set of ..... that computers use to perform specific operations or to exhibit desired behaviors.
2. A ..... contains a series of Visual Basic statements that perform an operation or calculate a value.

1.2 Object-oriented Programming in VB.NET

Before moving to the language syntax, let's formally define the key OO concepts and terms that will be used in this unit beginning with encapsulation, polymorphism, and inheritance.

1.2.1 Encapsulation

Encapsulation means that an object can hide its internal data structures from consumers of the object. Therefore, all of the object's internal data is manipulated through members (methods, properties, events, fields) of the object, rather than through direct references.

The primary benefits of encapsulation are maintainability and reusability. Code that takes advantage of encapsulation is more maintainable because consumers of the code work with the object through its public members. With a fully encapsulated object, for example, code outside the object cannot directly change a variable declared inside the object. By shutting off this direct access, fewer bugs are introduced because consumers of the object cannot inadvertently change the state of an object at run-time.

Abstracting the internal data of the object from consumers also leads to greater reusability. This follows because encapsulation leads to fewer dependencies between the consumer and the class and fewer dependencies is a prerequisite for creating reusable software.

1.2.2 Polymorphism

The second characteristic of OO systems is polymorphism. This concept is defined as the ability to write code that treats objects as if they were the same when in fact they are different. In other words, polymorphism allows you to write code that is generic across a set of objects that provide the same public members. Underneath the covers, each object might be implemented differently. However, as far as the consumer is concerned, each object looks the same and can be treated as such. In VB.NET, polymorphism can be created using both classes and interfaces.

The benefits of polymorphism revolve around the central fact that consumers of objects do not have to be aware of how the object performs its work, only that it does so through a specific set of members. This makes writing code that uses objects simpler by allowing the code to treat the

Notes

object as if it were a black box, which leads to increased maintainability. Along the same lines, polymorphism allows you to write less code because each individual object does not have to be dealt with separately. Finally, polymorphism lends itself to writing code that can be reused because it will not be specific to a particular object.

1.2.3 Inheritance

The final OO concept is inheritance. Inheritance allows objects to share their interfaces (the definition of their members) and/or implementation in a hierarchy. For example, Tyrannosaurus and Velociraptor objects might be derived or inherited from a more generic Theropod object. All three objects share a basic set of members and, possibly, behaviors, such as carnivorousness, although the descendant objects might also include additional members or override members of Theropod. Inheritance allows objects to become more specific further down the hierarchy by adding additional members. In a nutshell, inheritance allows objects to reuse features (either their definition or their code) of other objects to which they are naturally related. The primary benefit of inheritance is, thus, reuse.

Obviously, inheritance and polymorphism are closely related, and, in fact, inheritance is what makes polymorphism possible in OO designs. It is always the case that objects that are in an inheritance relationship can be treated polymorphically. For example, if the Velociraptor object is inherited from the Theropod object, any consumer that is designed to work with Theropod objects will also work with Velociraptor objects.

VB.NET developers can benefit from inheritance in two ways: through interface inheritance and implementation inheritance. Interface inheritance allows only the definition of the object to be reused, whereas implementation inheritance allows the actual code written for the ancestor object (and its ancestors all the way down the line) to be reused.

Note If developers wanted to use a form of implementation inheritance in previous versions of VB, they had to design classes to take advantage of the concepts of containment and delegation. Basically, these concepts mean that a class accessible by a consumer will contain a private instance of a second class and delegate its services to the consumer through its own interface. Containment and delegation are familiar terms (along with aggregation) to COM programmers.

Self Assessment

Fill in the blanks:

3. ..... means that an object can hide its internal data structures from consumers of the object.
4. ..... allows you to write code that is generic across a set of objects that provide the same public members.
5. ..... allows objects to share their interfaces (the definition of their members) and/or implementation in a hierarchy.

1.3 Visual Basic .NET

Visual Basic .NET is an Object-oriented programming language designed by Microsoft. With the word “Basic” being in the name of the language, you can already see that this is a language for

Notes

beginners. There are people who criticize VB.NET because of the simplicity of the syntax, but VB.NET has the ability to create very powerful and sophisticated applications. VB.NET is a great place to start because of how easy and straight forward it is. The syntax is easy and you will not find yourself writing hundreds of lines of code as there are many shortcuts that make coding so much easier in this language.

Let's take a look at the VB.NET syntax. The purpose of typing code is to instruct the application what to do. It's not as easy as typing "Hey application, multiply 5 by 83 but it's pretty darn close! If you wanted to tell your application to show a Message Box telling you that HowToStartProgramming.com is awesome, this would be the code you would use:

MessageBox.Show("HowToStartProgramming.com is awesome")

Microsoft Visual Basic .NET is a programming environment used to create Graphical User Interface (GUI) applications for the Microsoft Windows family of operating systems. It usually ships in two types, either by itself or as part of Microsoft Visual Studio .NET. To use the lessons on this site, you must have installed either Microsoft Visual Basic .NET 2003 or Microsoft Visual Studio .NET 2003. After installing Microsoft Visual Studio .NET 2003, to use the programming environment, you must first open it. To do that, you would click Start -> (All) Programs -> Microsoft Visual Studio .NET 2003 -> Microsoft Visual Studio .NET 2003.

In the early days of Microsoft DOS, there was a language called Basic. It provided a simplified and easy way to create small applications using words very close to the English language. Since the language was easy, it became popular with the help of Microsoft operating systems gaining ground. To continue this tendency and provide more support for Basic, Microsoft used that language as the platform to create Graphical User Interface (GUI) applications. Once again, this move was welcomed and the language became the widely accepted Visual Basic. The Microsoft Visual Basic programming environment became very popular for its ease of use and it was a candidate for serious productive applications. Because Visual Basic was not tied to the operating systems low-level operations, its programmers had to use library calls to access functions of the Win32 Application Programming Interface (API), the library that "defines" Microsoft Windows. This also accentuated the difference with other programming environments like Microsoft Visual C++ or other libraries like Microsoft Foundation Class (MFC). In fact, although Microsoft shipped Visual Studio 6 that combined various programming environments with different languages (Visual Basic, C++, ASP, Win32, etc), the only real thing they had in common was that they shipped in the same box (and the same DVD). To unify the various languages or programming environments that Microsoft had developed for many years, the company created a new library aside from Win32. This was the birth of the .NET Framework. This library is used by, or shared among, different programming languages or environments so that programmers can benefit from a better collaboration. Now it is possible for people who "speak", that is, people who program in, different languages to work on the same project with less regard for compatibility issues. This is because (most of) the functions, classes, and resources, are used in conceptually the same way in the different languages. Microsoft Visual Basic .NET is Microsoft's implementation of the .NET Framework for Visual Basic programmers. Although Visual Basic .NET is a "child" of Visual Basic 6.0, there are many differences that can be interpreted as a complete shift, with a lot of improvements. Because of these differences, many already Visual Basic 6.0 programmers resisted the move to this new environment (there were also many other considerations) but those programmers are catching up.

1.3.1 Common Language Runtime

The CLR is the execution engine for the .NET Framework. This runtime manages all code compiled with VB.NET. In fact, code compiled to run under .NET is called managed code to distinguish it from code running outside of the framework. Besides being responsible for

Notes

application loading and execution, the CLR provides services that will benefit component developers:

- Invocation and termination of threads and processes
- Object lifetime and memory management
- Cross-language integration
- Code access and role-based security
- Exception handling (even across languages)
- Deployment and versioning
- Interoperation between managed and unmanaged code
- Debugging and profiling support (even across languages)

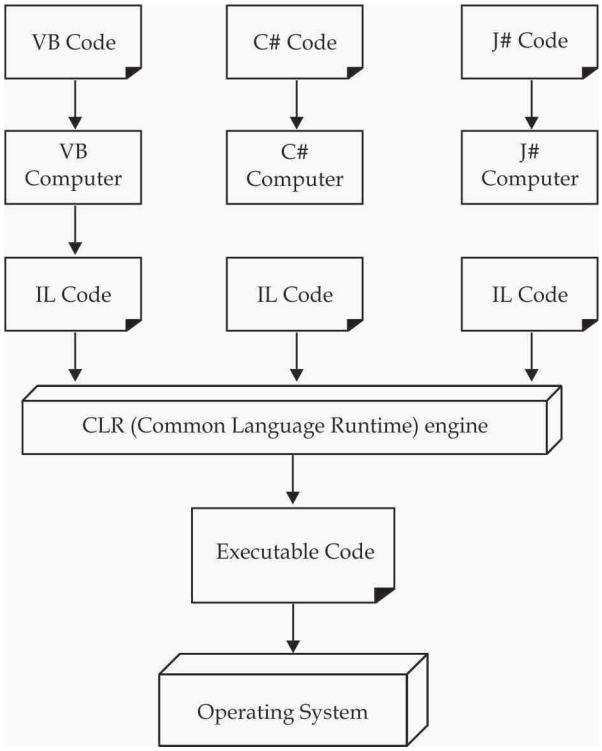
Runtimes are nothing new. Visual Basic has always had some form of a runtime. Visual C++ has a runtime called MSVCRT.DLL. Perl, Python, and SmallTalk also use runtimes. The difference between these runtimes and the CLR is that the CLR is designed to work with multiple programming languages. Every language whose compiler targets the .NET Framework benefits from the services of the CLR as much as any other language..NET is also similar to Java. Java uses a runtime called the Java Virtual Machine. It can run only with Java code, so it has the same limitations as the other languages. Another distinction is that the JVM is an interpreter. Although all languages in the .NET environment are initially compiled to a CPU independent language called Intermediate Language (which is analogous to Java byte code), IL is not interpreted at runtime like Java. When code is initially executed, one of several just-in-time (JIT) compilers translate the IL to native code on a method-by-method basis. Cross-language integration is one of the major benefits provided by the CLR. If a colleague has written a base class in C#, you can define a class in VB.NET that derives from it. This is known as cross-language inheritance. Also, objects written in different languages can easily interoperate. The two parts of the CLR that make this interoperation possible are the Common Type System and the Common Language Specification. The Common Language Runtime (CLR) is the virtual machine component of Microsoft's .NET framework and is responsible for managing the execution of .NET programs. In a process known as Just-in-time compilation, the compiled code is converted into machine instructions that, in turn, are executed by the computer's CPU. The CLR provides additional services including memory management, type safety and exception handling. All programs written for the .NET framework, regardless of programming language, are executed by the CLR. It provides exception handling, garbage collection and thread management. CLR is common to all versions of the .NET framework. The CLR is Microsoft's implementation of the Common Language Infrastructure (CLI) standard.

The Common Type System (CTS) defines rules that a language must adhere to in order to participate in the .NET Framework. It also defines a set of common types and operations that exist across most programming languages and specifies how these types are used and managed within the CLR, how objects expose their functionality and how they interoperate. The CTS forms the foundation that enables cross-language integration within .NET.

The Common Language Specification (CLS) is a subset of the CTS that describes the basic qualities used by a wide variety of languages. Components that use only the features of the CLS are said to be CLS-compliant. As a result, these components are guaranteed to be accessible from any other programming language that targets .NET. Because VB.NET is a CLS-compliant language, any class, object, or component that you build will be available from any other CLS-compliant programming language in .NET.

Notes

Figure 1.2: CLR



Source: http://dotnetslackers.com/articles/sql/Introduction-to-CLR-Database-Objects.aspx

1.3.2 Managed Execution

The .NET CLR provides a common context within which all .NET applications execute, regardless of the language in which they are written. CLR is responsible for handling every aspects of the managed code such as memory and resource management, secure environment to run in, garbage collection , access to the operating systems services etc. Code that targets the CLR is commonly known as *managed code.* The managed execution process includes the following steps:

1. Choosing a proper compiler
2. Generating MSIL code
3. Compiling MSIL to CPU specific native code using JIT
4. Executing the processor specific code.

Every constructs (such as class, struct, etc.) in every .NET languages must compile to CLR compatible types to qualify as .NET managed code. You can choose compilers such as Visual Basic, C#, Visual C++, JScript, or one of many third-party compilers like Eiffel, Perl, or COBOL compiler. CLR supports a wide variety of data types and language features. It is not mandatory to include all the CLR features in every .NET enabled languages, but the exposed language features should be compatible with the standard .NET frame work. If your component is targeted to use by components written in other .NET languages, your component's exported types must expose only language features that are included in the Common Language Specification (CLS).

1.3.3 Microsoft Intermediate Language (MSIL)

MSIL Code Generation is the first level of .NET compilation in which the high-level compiled in to a language called Intermediate Language (IL). The IL code look more like machine code than

Notes

high-level language, but the IL does contain some abstract concepts such as base classes and exception handling, which is why the language is called intermediate. MSIL includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations. When a compiler produces MSIL, it also produces metadata. Metadata describes the types in your code, including the definition of each type, the signatures of each type's members, the members that your code references, and other data that the runtime uses at execution time.

1.3.4 Just-In-Time Compiler

CPU-independent MSIL code can be efficiently converted to native code using Just-in-Time (JIT) compiler, only when that portion of IL code is required for execution. JIT generated machine code is CPU-specific code that runs on the same computer architecture as the JIT compiler and it takes advantage of the added instruction sets offered by each CPU type.

1.3.5 Executing Code

When CLR executes a .NET method for the first time, it generates a processor specific native code from MSIL using the JIT compiler. The next time the method is run, the existing JIT-compiled native code is run. The process of JIT-compiling and then executing the code is repeated until execution is complete. CLR provides myriad set of services to managed components like garbage collection, versioning, interoperability with unmanaged code, etc.

1.3.6 Assemblies

An assembly is a collection of types and resources that forms a logical unit of functionality. All types in the .NET Framework must exist in assemblies; the common language runtime does not support types outside of assemblies. Each time you create a Microsoft Windows® Application, Windows Service, Class Library, or other application with Visual Basic .NET, you're building a single assembly. Each assembly is stored as an .exe or .dll file.

Note Although it's technically possible to create assemblies that span multiple files, you're not likely to use this technology in most situations.

The .NET Framework uses assemblies as the fundamental unit for several purposes:

- Security
- Type Identity
- Reference Scope
- Versioning
- Deployment

1.3.7 Assembly Manifest

Every assembly contains an assembly manifest, a set of metadata with information about the assembly. The assembly manifest contains these items:

- The assembly name and version
- The culture or language the assembly supports (not required in all assemblies)

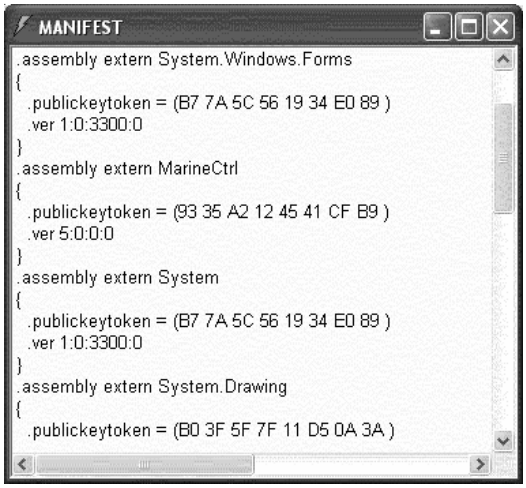
Notes

- The public key for any strong name assigned to the assembly (not required in all assemblies)
- A list of files in the assembly with hash information
- Information on exported types
- Information on referenced assemblies

In addition, you can add other information to the manifest by using assembly attributes. Assembly attributes are declared inside of a file in an assembly, and are text strings that describe the assembly. For example, you can set a friendly name for an assembly with the AssemblyTitle attribute:

```
<Assembly: AssemblyTitle("Test Project")>
```

Figure 1.3: Assembly Manifest



Source: http://msdn.microsoft.com/en-IN/library/ms973843.aspx

1.3.8 An End to DLL Hell

DLL hell is a common term for various problems associated with the use of dynamic link libraries (DLLs) or DLL files. A DLL file is a resource within the Windows operating system that contains code and data related to the functionality of one or more applications. These files, which may have the file extension .dll or other file extensions, have been a major building block for the Windows operating system and Windows programs since the early MS-DOS versions of Microsoft's computer technology. Successive versions of Windows have illustrated certain problems with the use of DLL files for many different programs. From a customer perspective, the most common versioning problem is what we call DLL Hell. Simply stated, DLL Hell refers to the set of problems caused when multiple applications attempt to share a common component like a Dynamic-Link Library (DLL) or a Component Object Model (COM) class. In the most typical case, one application will install a new version of the shared component that is not backward compatible with the version already on the machine. Although the application that has just been installed works fine, existing applications that depended on a previous version of the shared component might no longer work. In some cases, the cause of the problem is even more subtle. For example, consider the scenario where a user downloads a Microsoft ActiveX control as a side effect of visiting some Websites. When the control is downloaded it will replace any existing versions of the control that were present on the machine. If an application that has been installed on the machine happens to use this control, it too might potentially stop working. In many cases there is a significant delay before a user discovers that an application has stopped working. As a result, it is often difficult to remember when a change was made to the machine

Notes

that could have affected the application. A user may remember installing something a week ago, but there is no obvious correlation between that installation and the behavior they are now seeing. To make matters worse, there are few diagnostic tools available today to help the user (or the support person who is helping them) determine what is wrong. The reason for these issues is that version information about the different components of an application aren't recorded or enforced by the system. Also, changes made to the system on behalf of one application will typically affect all applications on the machine–building an application today that is completely isolated from changes is not easy. One reason why it's hard to build an isolated application is that the current run-time environment typically allows the installation of only a single version of a component or an application. This restriction means that component authors must write their code in a way that remains backward compatible, otherwise they risk breaking existing applications when they install a new component. In practice, writing code that is forever backward compatible is extremely difficult, if not impossible. In .NET, the notion of side-by-side is core to the versioning story. Side-by-side is the ability to install and run multiple versions of the same component on the machine at the same time. With components that support side-by-side, authors aren't necessarily tied to maintaining strict backward compatibility because different applications are free to use different versions of a shared component.

1.3.9 Global Assembly Cache (GAC)

Each computer where the common language runtime is installed has a machine-wide code cache called the global assembly cache. The global assembly cache stores assemblies specifically designated to be shared by several applications on the computer. You should share assemblies by installing them into the global assembly cache only when you need to. As a general guideline, keep assembly dependencies private, and locate assemblies in the application directory unless sharing an assembly is explicitly required. In addition, it is not necessary to install assemblies into the global assembly cache to make them accessible to COM interop or unmanaged code.

Note There are scenarios where you explicitly do not want to install an assembly into the global assembly cache. If you place one of the assemblies that make up an application in the global assembly cache, you can no longer replicate or install the application by using the **xcopy** command to copy the application directory. You must move the assembly in the global assembly cache as well.

There are several ways to deploy an assembly into the global assembly cache:

- Use an installer designed to work with the global assembly cache. This is the preferred option for installing assemblies into the global assembly cache.
- Use a developer tool called the Global Assembly Cache tool (Gacutil.exe), provided by the .NET Framework SDK.
- Use Windows Explorer to drag assemblies into the cache.

Note In deployment scenarios, use Windows Installer 2.0 to install assemblies into the global assembly cache. Use Windows Explorer or the Global Assembly Cache tool only in development scenarios, because they do not provide assembly reference counting and other features provided when using the Windows Installer.

Administrators often protect the WINNT directory using an Access Control List (ACL) to control write and execute access. Because the global assembly cache is installed in the WINNT directory,

Notes

it inherits that directory's ACL. It is recommended that only users with Administrator privileges be allowed to delete files from the global assembly cache. Assemblies deployed in the global assembly cache must have a strong name. When an assembly is added to the global assembly cache, integrity checks are performed on all files that make up the assembly. The cache performs these integrity checks to ensure that an assembly has not been tampered with, for example, when a file has changed but the manifest does not reflect the change.

Self Assessment

True or False:

6. The CTS is the execution engine for the .NET Framework.
7. The code compiled to run under .NET is called unmanaged code.
8. MSIL Code Generation is the first level of .NET compilation in which the high-level compiled in to a language called intermediate language (IL).
9. CPU-independent MSIL code can be efficiently converted to native code using an interpreter.

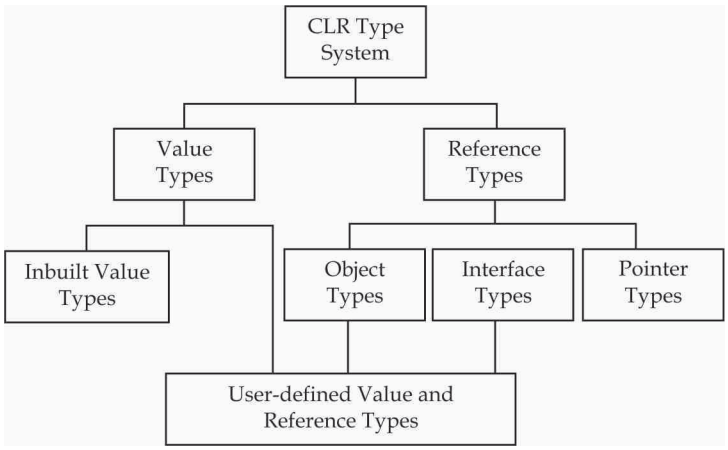
1.4 The Common Type System

The Common Type System defines how data types are declared, used, and managed in the runtime, and is also an important part of the runtime's support for the Cross Language Integration. The common type system performs the following functions:

- Establishes a framework that enables cross-language integration, type safety, and high performance code execution.
- Provides an object-oriented model that supports the complete implementation of many programming languages.
- Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.

The Common Type System can be divided into two general categories of types–Reference type and Value type each of which is further divided into subcategories.

Figure 1.4: CTS



Source: http://msdn.microsoft.com/en-us/library/ms973862.aspx

Notes

1.4.1 Classes

Class defines the operations an object can perform (methods, events, or properties) and defines a value that holds the state of the object (fields). Although a class generally includes both definition and implementation, it can have one or more members that have no implementation. An instance of a class is an object. You access an object's functionality by calling its methods and accessing its properties, events, and fields.

The Table 1.1 provides a description of some of the characteristics that the runtime allows a class to have. (Additional characteristics that are available through Attribute classes are not included in this list.) Your language might not make all these characteristics available.

Table 1.1: Characteristics of a Class

Characteristic	Description
Sealed	Specifies that another type cannot be derived from this type.
Implements	Indicates that the class uses one or more interfaces by providing implementations of interface members.
Abstract	Specifies that you cannot create an instance of the class. To use it, you must derive another class from it.
Inherits	Indicates that instances of the class can be used anywhere the base class is specified. A derived class that inherits from a base class can use the implementation of any virtual methods provided by the base class, or derived class can override them with its own implementation.
Exported or not exported	Indicates whether a class is visible outside the assembly in which it is defined. Applies only to top-level classes.

Source: http://msdn.microsoft.com/en-us/library/2s9w552e%28v=vs.71%29.aspx

Nested classes also have member characteristics. Class members that have no implementation are abstract members. A class that has one or more abstract members is itself abstract; new instances of it cannot be created. Some languages that target the runtime allow you to mark a class as abstract even if none of its members are abstract. You can use an abstract class when you need to encapsulate a basic set of functionality that derived classes can inherit or override when appropriate. Classes that are not abstract are referred to as concrete classes. A class can implement any number of interfaces, but it can inherit from only one base class. All classes must have at least one constructor, which initializes new instances of the class. Each language that supports the runtime provides a way to indicate that a class or class member has specific characteristics. When you use the syntax required by your language, the language ensures that the characteristics of the class and its members are stored (as metadata) along with the implementation of the class.

1.4.2 Interfaces

An interface is basically a class definition. It itself cannot be instantiated, rather it is implemented by other classes. Interfaces can be defined and implemented in VB6 albeit with some workarounds and limitations. For example, in VB6 you cannot use one variable alone to access the methods of the interface and of the implementing class–you have to declare two variables, one as the interface and the other as the implementing class and point both of them to the same instance. VB.NET has a different and more straightforward implementation of interfaces. Like classes, interfaces can be declared in any class file and are declared with a Interface End Interface statement block:

Notes

Example:

```
Interface IAudit
    Public orderID as long
    Public Function Log (Msg as String)
End Interface
```

1.4.3 Value Types

The common type system supports two general categories of types:

Value types: They directly contain their data, and instances of value types are either allocated on the stack or allocated in line in a structure. Value types can be built-in (implemented by the runtime), user-defined, or enumerations.

Reference types : They store a reference to the value's memory address, and are allocated on the heap. Reference types can be self-describing types, pointer types, or interface types. The type of a reference type can be determined from values of self-describing types. Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types, and delegates.

Example:

```
Imports System

Class Class1
    Public Value As Integer = 0
End Class 'Class1

Class Test
    Shared Sub Main()
        Dim val1 As Integer = 0
        Dim val2 As Integer = val1
        val2 = 123
        Dim ref1 As New Class1()
        Dim ref2 As Class1 = ref1
        ref2.Value = 123
        Console.WriteLine("Values: {0}, {1}", val1, val2)
        Console.WriteLine("Refs: {0}, {1}", ref1.Value, ref2.Value)
    End Sub 'Main
End Class 'Test
Output:
Values: 0, 123
Refs: 123, 123
```

1.4.4 Delegates

The runtime supports reference types called delegates that serve a purpose similar to that of function pointers in C++. Unlike function pointers, a delegate instance is independent of the classes of the methods it encapsulates; all that matters is that those methods be compatible with the delegate's type. Also, while function pointers can only reference static functions, a delegate can reference both static and instance methods. Delegates are mainly used for event handlers

Notes

and callback functions in the .NET Framework. All delegates inherit from System.Delegate, and have an invocation list, which is a linked list of methods that are executed when the delegate is invoked. The resulting delegate can reference any method with a matching signature. The return value is not defined for a delegate that has a return type and contains more than one method in its invocation list. You can use the delegate's Combine and Remove methods to add and remove methods to its invocation list. To call the delegate, use the Invoke method, or the BeginInvoke and EndInvoke methods to call the delegate asynchronously. The implementations of the delegate class are provided by the runtime, not by user code.

Self Assessment

True or False:

10. An interface is basically a class definition which cannot be instantiated.

11. Value types are allocated on the heap.

1.5 Features of VB.NET

Visual Basic .NET (VB.NET) is an object-oriented computer programming language implemented on the .NET Framework. Although it is an evolution of classic Visual Basic language, it is not backwards-compatible with VB6, and any code written in the old version does not compile under VB.NET. Like all other .NET languages, VB.NET has complete support for object-oriented concepts. Everything in VB.NET is an object, including all of the primitive types (Short, Integer, Long, String, Boolean, etc.) and user defined types, events, and even assemblies. All objects inherits from the base class Object. VB.NET is implemented of Microsoft's .NET framework. Therefore it has full access all the libraries in the .Net Framework. It's also possible to run VB.NET programs on Mono, the open-source alternative to .NET, not only under Windows, but even Linux or Mac OSX.

The following reasons make VB.Net a widely used professional language:

- Modern, general purpose.
- Object-oriented.
- Component-oriented.
- Easy to learn.
- Structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- Part of .Net Framework.

VB.NET has numerous strong programming features that make it endearing to multitude of programmers worldwide. Let us mention some of these features:

- Boolean Conditions
- Automatic Garbage Collection
- Standard Library
- Assembly Versioning
- Properties and Events

Notes

- Delegates and Events Management
- Easy to use Generics
- Indexers
- Conditional Compilation
- Simple Multithreading

1.6 .NET Framework

The .NET Framework (pronounced *dot net*) is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It includes a large library and provides language interoperability (each language can use code written in other languages) across several programming languages. Programs written for the .NET Framework execute in a software environment (as contrasted to hardware environment), known as the Common Language Runtime (CLR), an application virtual machine that provides services such as security, memory management, and exception handling. The class library and the CLR together constitute the .NET Framework. The .NET Framework's Base Class Library provides user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. Programmers produce software by combining their own source code with the .NET Framework and other libraries. The .NET Framework is intended to be used by most new applications created for the Windows platform. Microsoft also produces an integrated development environment largely for .NET software called Visual Studio.

1.6.1 Visual Basic in .NET Framework

Visual Basic .NET is designed around the .NET Framework, which provides enhanced security, memory management, versioning, and deployment support. The .NET Framework also enables interoperability between objects you create with any .NET programming language. This means you can create objects with Visual Basic .NET that are easy to use from other .NET languages, and you can use objects from other .NET languages just like you use objects created with Visual Basic .NET.

1.7 VB.NET as a Language in .NET Framework

We will now discuss the features of VB.NET as a language and its basic constructs.

1.7.1 Source Files

The files of a VB.NET project are stored with a *.vb* extension. If we include Visual Basic .NET code in ASP.NET web page files, then we keep the extension as .aspx, Source file is any collection of computer instructions written using some human-readable computer language, usually as text. The source code of a program is specially designed to facilitate the work of computer programmers, who specify the actions to be performed by a computer mostly by writing source code. Visual Studio .NET source files are kept in the Solution Explorer window, and all source is included from these files when the solution is built.

1.7.2 Identifiers

An *identifier* is a name. Visual Basic .NET identifiers conform to the Unicode Standard Annex 15 with one exception: identifiers may begin with an underscore (connector) character. If an identifier

Notes

begins with an underscore, it must contain at least one other valid identifier character to disambiguate it from a line continuation. Regular identifiers may not match keywords, but escaped identifiers can. An *escaped identifier* is an identifier delimited by square brackets. Escaped identifiers follow the same rules as regular identifiers except that they may match keywords and may not have type characters.

This example defines a class named class with a shared method named shared that takes a parameter named boolean and then calls the method.

```
Imports System

Class [class]
    Shared Sub [shared](ByVal [boolean] As Boolean)
        If [boolean] Then
            Console.WriteLine("true")
        Else
            Console.WriteLine("false")
        End If
    End Sub
End Class

Module [module]
    Sub Main()
        [class].[shared](True)
    End Sub
End Module
```

Identifiers are case insensitive, so two identifiers are considered to be the same identifier if they differ only in case.

> *Note* The Unicode Standard one-to-one case mappings are used when comparing identifiers, and any locale-specific case mappings are ignored.

1.7.3 Keywords

A *keyword* is a word that has special meaning in a language construct. All keywords are reserved by the language and may not be used as identifiers unless the identifiers are escaped.

> *Note* EndIf, GoSub, Let, Variant, and Wend are retained as keywords, although they are no longer used in Visual Basic .NET.

Its syntax is,

```
Keyword ::= < member of keyword table >
```

Notes

A list of common keywords is summarized in the Table 1.2.

Table 1.2: Keywords of VB.NET

AddHandler	AddressOf	Alias	And
AndAlso	Ansi	As	Assembly
Auto	Booleam	ByRef	Byte
ByVal	Call	Case	Catch
CBool	CByte	CChar	CDate
CDbl	CDec	Char	CInt
Class	CLng	CObj	Const
CShort	CSng	CStr	CType
Date	Decimal	Declare	Default
Delegate	Dim	DirectCast	Do
Double	Each	Else	Elself
End	EndIf	Enum	Erase
Error	Event	Exit	False
Finally	For	Friend	Function
Get	GetType	GoSub	GoTo
Handles	If	Implements	Imports
In	Inherits	Integer	Interface
Is	Let	Lib	Like
Long	Loop	Me	Mod
Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	New	Next
Not	Nothing	NotInheritable	NotOverridable
Object	On	Option	Optional
Or	OrElse	Overloads	Overridable
Overrides	ParamArray	Preserve	Private
Property	Protected	Public	RaiseEvent
ReadOnly	ReDim	REM	RemoveHandler
Resume	Return	Select	Set
Shadows	Shared	Short	Single
Static	Step	Stop	String
Structure	Sub	SyncLock	Then
Throw	To	True	Try

Source: http://msdn.microsoft.com/en-IN/library/aa711646%28v=vs.71%29.aspx

1.7.4 Literals

A *literal* is a textual representation of a particular value of a type. Literal types include Boolean, integer, floating point, string, character, and date.

```
Literal ::=
   BooleanLiteral |
```

Notes

```
    IntegerLiteral |
    FloatingPointLiteral |
    StringLiteral |
    CharacterLiteral |
    DateLiteral |
    Nothing
```

Boolean Literals: True and False are literals of the Boolean type that map to the true and false state, respectively.

```
BooleanLiteral ::= True | False
```

Integer Literals: Integer literals can be decimal (base 10), hexadecimal (base 16), or octal (base 8). A decimal integer literal is a string of decimal digits (0-9). A hexadecimal literal is & H followed by a string of hexadecimal digits (0-9, A-F). An octal literal is & O followed by a string of octal digits (0-7). Decimal literals directly represent the decimal value of the integral literal, whereas octal and hexadecimal literals represent the binary value of the integer literal (thus, &H8000S is –32768, not an overflow error). The type of a literal is determined by its value or by the following type character. If no type character is specified, values in the range of the Integer type are typed as Integer; values outside the range for Integer are typed as Long. If an integer literal's type is of insufficient size to hold the integer literal, a compile-time error results.

Floating-point Literals: A floating-point literal is an integer literal followed by an optional decimal point (the ASCII period character) and mantissa, and an optional base 10 exponent. By default, a floating-point literal is of type Double. If the Single, Double, or Decimal type character is specified, the literal is of that type. If a floating-point literal's type is of insufficient size to hold the floating-point literal, a compile-time error results.

String Literals: A string literal is a sequence of zero or more Unicode characters beginning and ending with an ASCII double-quote character, a Unicode left double-quote character, or a Unicode right double-quote character. Within a string, a sequence of two double-quote characters is an escape sequence representing a double quote in the string. A string constant is of the String type.

***Character Literals*:** A character literal represents a single Unicode character of the Char type. Two double-quote characters is an escape sequence representing the double-quote character.

Example:

```
Module Test
    Sub Main()
        ' This prints out: a.
        Console.WriteLine("a"c)

        ' This prints out: ".
        Console.WriteLine(""""c)
    End Sub
End Module
```

Date Literals: A date literal represents a particular moment in time expressed as a value of the Date type. The literal may specify both a date and a time, just a date, or just a time. If the date value is omitted, then January 1 of the year 1 in the Gregorian calendar is assumed. If the time value is omitted, then 12:00:00 AM is assumed. To avoid problems with interpreting the year value in a date value, the year value cannot be two digits. When expressing a date in the first century AD/CE, leading zeros must be specified. A time value may be specified either using a 24-hour value or a 12-hour value; time values that omit an AM or PM are assumed to be 24-hour values. If a time value omits the minutes, the literal 0 is used by default. If a time value omits the

Notes

seconds, the literal 0 is used by default. If both minutes and second are omitted, then AM or PM must be specified. If the date value specified is outside the range of the Date type, a compile-time error occurs.

Nothing: Nothing is a special literal; it does not have a type and is convertible to all types in the type system. When converted to a particular type, it is the equivalent of the default value of that type.

```
Nothing ::= Nothing
```

1.7.5 Types

Arrays: An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

To declare an array in VB.NET, you use the Dim statement.

Example:

```
Dim intData(30)                    ' an array of 31 elements
Dim strData(20) As String          ' an array of 21 strings
Dim twoDarray(10, 20) As Integer   'a two dimensional array of integers
Dim ranges(10, 100)                'a two dimensional array
```

Collections: Collection classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. Most collection classes implement the same interfaces. Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc. These classes create collections of objects of the Object class, which is the base class for all data types.

Example:

ArrayList class represents an ordered collection of an object that can be indexed individually.

Case Study

Very Busy (VB) Mail Order

If you don't have the time to look for all those hard-to-find items, tell us what you're looking for. We'll send you a catalog from the appropriate company or order for you. We can place an order and ship it to you. We also help with shopping for gifts; your order can be gift wrapped and sent anywhere you wish.

The company title will be shortened to VB Mail Order. Include this name on the title bar of the first form of each project that you create for this case study.

Your first job is to create a project for VB Mail Order that will display the typical shipping times from some of our most popular catalogs. When a customer makes an order from us, we order the products from the manufacturer. Once the products are received in our warehouse it takes one day to gift-wrap (if necessary), box up the product and ship it.

Contd...

Notes

We can send our products to the customer via Ship-It Ground (five day) or, for an extra $10, Express Overnight (next day).

Include a button for each catalog. When the user clicks on the button for a catalog, display the shipping time in a label. Also include an identifying label "Time Until at Customer:".

Be sure to include a button for Exit.

Include a label at the bottom of the form that holds "Programmed by " and your name.

Test Data

Table 1

Catalog	Label Text
Odds and Ends	Ships each week on Tuesday and takes three days to arrive. If the customer orders on Wednesday for ground shipping then the order will arrive in 16 days (the maximum amount of time). If the customer orders on Tuesday for express shipping then the order will arrive in six days (the minimum amount of time).
Camping Needs	They will ship directly to the customer in two days so, if the customer orders by Ship-It, then the order will arrive in seven days. If the order is shipped via Express Overnight then the order will be there in three days.
The Outlet	Ships everyday of the week and arrives at our warehouse in two days (three days total). If the customer chooses ground shipping, their package will arrive in eight days. By express the package will get there in four days.

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://highered.mcgraw-hill.com/sites/dl/free/0072459034/47748/bradley_ch01.pdf

Self Assessment

Fill in the blanks:

12. ..... classes are specialized classes for data storage and retrieval.
13. An ..... is a name.
14. A ..... is a word that has special meaning in a language construct.
15. A ..... is a textual representation of a particular value of a type.

1.8 Summary

- Computer programming, often shortened to programming, scripting, or coding is the process of designing, writing, testing, debugging, and maintaining the source code of computer programs.
- The two basic elements of Visual Basic are the GUI and the code associated with the application that makes it respond to events occurred as a result of a user action.
- IDE is a term commonly used in the programming world to describe the interface and environment that we use to create our applications.

Notes

- A procedure contains a series of Visual Basic statements that perform an operation or calculate a value.
- Encapsulation means that an object can hide its internal data structures from consumers of the object.
- Polymorphism allows you to write code that is generic across a set of objects that provide the same public members.
- Inheritance allows objects to share their interfaces (the definition of their members) and/or implementation in a hierarchy.
- Visual Basic .NET is an Object-oriented programming language designed by Microsoft.
- The CLR is the execution engine for the .NET Framework.
- Code compiled to run under .NET is called managed code to distinguish it from code running outside of the framework.
- The Common Language Specification (CLS) is a subset of the CTS that describes the basic qualities used by a wide variety of languages.
- MSIL Code Generation is the first level of .NET compilation in which the high-level compiled in to a language called intermediate language (IL).
- CPU-independent MSIL code can be efficiently converted to native code using Just-in-Time (JIT) compiler.
- An assembly is a collection of types and resources that forms a logical unit of functionality.
- A DLL file is a resource within the Windows operating system that contains code and data related to the functionality of one or more applications.
- The global assembly cache stores assemblies specifically designated to be shared by several applications on the computer.
- The Common Type System defines how data types are declared, used, and managed in the runtime, and is also an important part of the runtime's support for the Cross Language Integration.

1.9 Keywords

Assembly: It is a collection of types and resources that forms a logical unit of functionality.

CLR: It is the execution engine for the .NET Framework.

Common Language Specification: It is a subset of the CTS that describes the basic qualities used by a wide variety of languages.

Encapsulation: It means that an object can hide its internal data structures from consumers of the object.

IDE: It is a term commonly used in the programming world to describe the interface and environment that we use to create our applications.

Inheritance: It allows objects to share their interfaces (the definition of their members) and/or implementation in a hierarchy.

Polymorphism: It allows you to write code that is generic across a set of objects that provide the same public members.

Procedure: It contains a series of Visual Basic statements that perform an operation or calculate a value.

Notes

1.10 Review Questions

1. What are the two major features of Visual basic?
2. How can we say that VB.NET is object-oriented?
3. Write a short note on CLR.
4. What is managed code?
5. Why is JIT needed?
6. What do you mean by MSIL?
7. Explain the concept of assemblies.
8. What is CTS?
9. Differentiate between value and reference types.
10. Explain the different kinds of literals.

Answers: Self Assessment

1.	Instructions	2.	Procedure
3.	Encapsulation	4.	Polymorphism
5.	Inheritance	6.	False
7.	False	8.	True
9.	False	10.	True
11.	False	12.	Collection
13.	Identifier	14.	Keyword
15.	Literal		

1.11 Further Readings

Books

Beginning Vb.Net 2003, Willis.

Building Distributed Applications with Visual Basic.NET, Dan Fox, Sams.

Object-oriented Programming with Visual Basic.Net, Hamilton.

Programming Visual Basic .NET, J. Liberty.

VB .NET Language in a Nutshell, Steven Roman, Ron Petrusha, Paul Lomax.

Visual Basic.NET Black Book, Steven Holzner.

Online links

http://www.visualbasicbooks.com/progVB6samplepg6.html

http://howtostartprogramming.com/vb-net/

http://www.homeandlearn.co.uk/net/nets1p1.html

http://www.jblearning.com/samples/0763724785/ch02_bronson.pdf

http://vb.net-informations.com/framework/framework_tutorials.htm

Notes

Unit 2: Variables and Data Types

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the operators available
- Understand the concept of variables

Introduction

In order to store values temporarily we make use of variables. Variables have a name that helps to refer to it and a data type that determines the kind of data the variable can store. Constants are used when we need the values to remain constant throughout the execution of an application. Using constants can make your code more readable by providing meaningful names instead of numbers. Data types control the internal storage of data in Visual Basic. By default, Visual Basic uses the Variant data type.

We will now discuss each of them in detail.

2.1 Variables

Variables are used to temporarily store values during the execution of an application. Variables have a name and a type. You can think of a variable as a placeholder in memory for an unknown value.

Notes

Example: Imagine you are creating a program for a fruit stand to track the sales of apples. You don't know the price of an apple or the quantity sold until the sale actually occurs. You can use two variables to hold the unknown values — let's name them ApplePrice and ApplesSold. Each time the program is run, the user supplies the values for the two variables. To calculate the total sales and display it in a Textbox named txtSales, your code would look like this:

```
txtSales.txt = ApplePrice * ApplesSold
```

The expression returns a different total each time, depending on what values the user provides. The variables allow you to make a calculation without having to know in advance what the actual inputs are.

In this example, the data type of ApplePrice is Currency; the data type of ApplesSold is an integer. Variables can represent many other values as well: text values, dates, various numeric types, even objects.

You use assignment statements to perform calculations and assign the result to a variable:

```
ApplesSold = 10 // The value 10 is passed to the variable.
ApplesSold = ApplesSold + 1 // The variable is incremented.
```

Note The equal sign in this example is an assignment operator, not an equality operator; the value (10) is being assigned to the variable (ApplesSold).

You declare a variable with the Dim statement, supplying a name for the variable:

Dim *variablename* [**As** *type*]

Variables declared with the Dim statement within a procedure exist only as long as the procedure is executing. When the procedure finishes, the value of the variable disappears. In addition, the value of a variable in a procedure is *local* to that procedure — that is, you can't access a variable in one procedure from another procedure. These characteristics allow you to use the same variable names in different procedures without worrying about conflicts or accidental changes.

A variable name:

- Must begin with a letter.
- Can't contain an embedded period or embedded type-declaration character.
- Must not exceed 255 characters.
- Must be unique within the same *scope,* which is the range from which the variable can be referenced — a procedure, a form, and so on.

The optional As *type* clause in the Dim statement allows you to define the data type or object type of the variable you are declaring. Data types define the type of information the variable stores. Some examples of data types include String, Integer, and Currency. Variables can also contain objects from Visual Basic or other applications. Examples of Visual Basic object types, or classes, include Object, Form1, and TextBox.

There are other ways to declare variables:

- Declaring a variable in the Declarations section of a form, standard, or class module, rather than within a procedure, makes the variable available to all the procedures in the module.

Notes

- Declaring a variable using the Public keyword makes it available throughout your application.
- Declaring a local variable using the Static keyword preserves its value even when a procedure ends.

In VB. NET you don't have to declare a variable before using it. For example, you could write a function where you don't need to declare TempVal before using it:

```
Function SafeSqr(num)
    TempVal = Abs(num)
    SafeSqr = Sqr(TempVal)
End Function
```

Visual Basic automatically creates a variable with that name, which you can use as if you had explicitly declared it. While this is convenient, it can lead to subtle errors in your code if you misspell a variable name. For example, suppose that this was the function you wrote:

```
Function SafeSqr(num)
    TempVal = Abs(num)
    SafeSqr = Sqr(TemVal)
End Function
```

At first glance, this looks the same. But because the TempVal variable was misspelled on the next-to-last line, this function will always return zero. When Visual Basic encounters a new name, it can't determine whether you actually meant to implicitly declare a new variable or you just misspelled an existing variable name, so it creates a new variable with that name.

To avoid the problem of misnaming variables, you can stipulate that Visual Basic always warn you whenever it encounters a name not declared explicitly as a variable. To explicitly declare variables, place this statement in the Declarations section of a class, form, or standard module:

```
Option Explicit
```

or

From the Tools menu, choose Options, click the Editor tab and check the Require Variable Declaration option. This automatically inserts the Option Explicit statement in any new modules, but not in modules already created; therefore, you must manually add Option Explicit to any existing modules within a project.

Had this statement been in effect for the form or standard module containing the SafeSqr function, Visual Basic would have recognized TempVal and TemVal as undeclared variables and generated errors for both of them. You could then explicitly declare TempVal:

```
Function SafeSqr(num)
    Dim TempVal
    TempVal = Abs(num)
    SafeSqr = Sqr(TemVal)
End Function
```

Now you'd understand the problem immediately because Visual Basic would display an error message for the incorrectly spelled TemVal. Because the Option Explicit statement helps you catch these kinds of errors, it's a good idea to use it with all your code.

Notes

Note The Option Explicit statement operates on a per-module basis; it must be placed in the Declarations section of every form, standard, and class module for which you want Visual Basic to enforce explicit variable declarations.

If you select Require Variable Declaration, Visual Basic inserts Option Explicit in all subsequent form, standard, and class modules, but does not add it to existing code. You must manually add Option Explicit to any existing modules within a project.

2.1.1 Variable Scope and Lifetime

The scope of a variable, sometimes referred to as accessibility of a variable, refers to where the variable can be read from and/or written to, and the variable's lifetime, or how long it stays in memory. The scope of a procedure or method refers to where a procedure can be called from or under what context you are allowed to call a method.

Let's take a quick look at the terms that you will need to know.

Table 2.1: Some Important Terms

Term	Used with…	Visibility
Public	Variables/Properties/Methods/Types	Anywhere in or outside of a project
Private	Variables/Properties/Methods/Types	Only in the block where defined
Protected	Variables/Properties/Methods	Can be used in the class where defined. Can be used within any inherited class.
Friend	Variables/Properties/Methods	Can only be assessed by code in the same project/assembly.
ProtectedFriend	Variables/Properties/Methods	Combination of Protected and Friend

Source: http://msdn.microsoft.com/en-us/library/ms973875.aspx

The scope of a variable defines which parts of your code are aware of its existence. When you declare a variable within a procedure, only code within that procedure can access or change the value of that variable; it has a scope that is local to that procedure. Sometimes, however, you need to use a variable with a broader scope, such as one whose value is available to all the procedures within the same module, or even to all the procedures in your entire application. Visual Basic allows you to specify the scope of a variable when you declare it.

Depending on how it is declared, a variable is scoped as either a procedure-level (local) or module-level variable.

Table 2.2: Scope of a Variable

Scope	Private	Public
Procedure-level	Variables are private to the procedure in which they appear.	Not applicable. You cannot declare public variables within a procedure.
Module-level	Variables are private to the module in which they appear.	Variables are available to all modules.

Source: http://msdn.microsoft.com/en-us/library/aa231205%28v=vs.60%29.aspx

Notes

Procedure-level Variables: They are recognized only in the procedure in which they're declared. These are also known as local variables. You declare them with the Dim or Static keywords.

Example:

Dim intTemp as Integer

or

Static intPermanent as Integer

Values in local variables declared with Static exist the entire time your application is running while variables declared with Dim exist only as long as the procedure is executing.

Local variables are a good choice for any kind of temporary calculation. For example, you can create a dozen different procedures containing a variable called intTemp. As long as each intTemp is declared as a local variable, each procedure recognizes only its own version of intTemp. Any one procedure can alter the value in its local intTemp without affecting intTemp variables in other procedures.

Variables Used within a Module: By default, a module-level variable is available to all the procedures in that module, but not to code in other modules. You create module-level variables by declaring them with the Private keyword in the Declarations section at the top of the module.

Example:

Private intTemp as Integer

At the module level, there is no difference between Private and Dim, but Private is preferred because it readily contrasts with Public and makes your code easier to understand.

Variables Used by all Modules: To make a module-level variable available to other modules, use the Public keyword to declare the variable. The values in public variables are available to all procedures in your application. Like all module-level variables, public variables are declared in the Declarations section at the top of the module.

Example:

Public intTemp as Integer

Note You can't declare public variables within a procedure, only within the Declarations section of a module.

The lifetime of a declared element is the period of time during which it is available for use. Variables are the only elements that have lifetime. For this purpose, the compiler treats procedure parameters and function returns as special cases of variables. The lifetime of a variable represents the period of time during which it can hold a value. Its value can change over its lifetime, but it always holds some value. A member variable (declared at module level, outside any procedure) typically has the same lifetime as the element in which it is declared. A non-shared variable declared in a class or structure exists as a separate copy for each instance of the class or structure in which it is declared. Each such variable has the same lifetime as its instance. However, a Shared variable has only a single lifetime, which lasts for the entire time your application is

Notes

running. A local variable (declared inside a procedure) exists only while the procedure in which it is declared is running. This applies also to that procedure's parameters and to any function return. However, if that procedure calls other procedures, the local variables retain their values while the called procedures are running.

A local variable's lifetime begins when control enters the procedure in which it is declared. Every local variable is initialized to the default value for its data type as soon as the procedure begins running. When the procedure encounters a Dim statement that specifies initial values, it sets those variables to those values, even if your code had already assigned other values to them. Each member of a structure variable is initialized as if it were a separate variable. Similarly, each element of an array variable is initialized individually. Variables declared within a block inside a procedure (such as a For loop) are initialized on entry to the procedure. These initialisations take effect whether or not your code ever executes the block.

2.1.2 Variables of Built-in Objects

We can also define a variable of the type object. Let us discuss how.

The Object data type can point to data of any data type, including any object instance your application recognizes. Use Object when you do not know at compile time what data type the variable might point to. The default value of Object is Nothing (a null reference). You can assign a variable, constant, or expression of any data type to an Object variable. To determine the data type an Object variable currently refers to, you can use the GetTypeCode method of the System.Type class.

Example:

```
Dim myObject As Object
' Suppose myObject has now had something assigned to it.
Dim datTyp As Integer
datTyp = Type.GetTypeCode(myObject.GetType())
```

The Object data type is a reference type. However, Visual Basic treats an Object variable as a value type when it refers to data of a value type. Whatever data type it refers to, an Object variable does not contain the data value itself, but rather a pointer to the value. It always uses four bytes in computer memory, but this does not include the storage for the data representing the value of the variable. Because of the code that uses the pointer to locate the data, Object variables holding value types are slightly slower to access than explicitly typed variables.

Application Object: It is used to refer to an application. Its syntax is,

```
Dim app_1 As Application
```

Example: To declare a variable that refers to a Microsoft Access database, the above declaration would be made as,

```
Dim app_1 As Access.Application
```

Database Object: It allows you to get a reference to the database you are using.

Example:

```
Dim curDatabase As Database
```

Notes

Self Assessment

Fill in the blanks:

1. ..... are used to temporarily store values during the execution of an application.
2. Variables have a ..... and a ..... .
3. Variables declared with the ..... statement within a procedure exist only as long as the procedure is executing.
4. The ..... of a variable defines which parts of your code are aware of its existence.
5. ..... are recognized only in the procedure in which they're declared.
6. To make a ..... -level variable available to other modules, use the Public keyword to declare the variable.

2.2 Operators

We will now discuss the various kinds of operators available in VB.NET.

2.2.1 Mathematical Expressions

Some of the mathematical operators used in VB.NET are summarized in Table 2.3.

Table 2.3: Mathematical Operators

Operator	Description
^	Raises one operand to the power of another
+	Adds two operands
–	Subtracts second operand from the first
*	Multiply both operands
/	Divide one operand by another and returns a floating point result
\	Divide one operand by another and returns an integer result
MOD	Modulus Operator and remainder of after an integer division

Source: http://www.tutorialspoint.com/vb.net/vb.net_operators.htm

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

Example: x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedence than + so it first get multiplied with 3*2 and then adds into 7.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Table 2.4: Operator Precedence

Operator	Precedence
Await	Highest
Exponentiation (^)	
Unary identity and negation (+, –)	

Contd...

Notes

Multiplication and floating-point divisioin (*, /)	
Integer division (\)	
Modulus arithmetic (Mod)	
Addition and subtraction (+, –)	
Arithmetic bit shift (<<, >>)	
All comparison operators (=, <>, <, <=, >, >=, Is, IsNot, Like, TypeOf…Is)	
Negation (Not)	
Conjunction (And, AndAlso)	
Inclusive disjunction (Or, OrElse)	
Exclusive disjunction (Xor)	Lowest

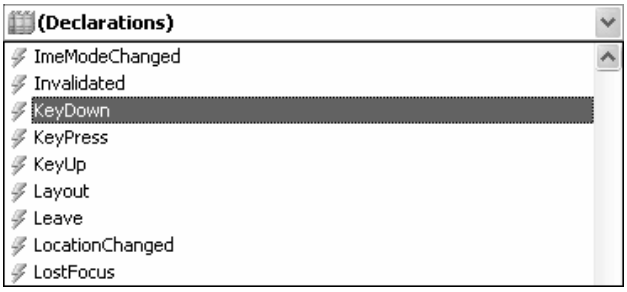
Source: http://www.tutorialspoint.com/vb.net/vb.net_operators.htm

2.2.2 Event-driven Programming

An event is a signal that informs an application that something important has occurred.

For example, when a user clicks a control on a form, the form can raise a **Click** event and call a procedure that handles the event. Events also allow separate tasks to communicate. For example, that your application performs a sort task separately from the main application. If a user cancels the sort, your application can send a cancel event instructing the sort process to stop.

Figure 2.1: Events



Source: http://www.homeandlearn.co.uk/net/nets10p3.html

2.2.3 Relational Operators

The *relational operators* compare values to one other. The comparison operators are =, <>, <, >, <=, and >=. All of the relational operators result in a Boolean value.

The relational operators have the following general meaning:

- The = operator tests whether the two operands are equal.
- The <> operator tests whether the two operands are not equal.
- The < operator tests whether the first operand is less than the second operand.
- The > operator tests whether the first operand is greater than the second operand.
- The <= operator tests whether the first operand is less than or equal to the second operand.
- The >= operator tests whether the first operand is greater than or equal to the second operand.

Notes

Logical Operators

The And, Not, Or, and Xor operators, which are called the logical operators, are evaluated as follows:

- For the Boolean type:
  - A logical And operation is performed on its two operands.
  - A logical Not operation is performed on its operand.
  - A logical Or operation is performed on its two operands.
  - A logical exclusive-Or operation is performed on its two operands.
- For Byte, Short, Integer, Long, and all enumerated types, the specified operation is performed on each bit of the binary representation of the two operand(s):
  - *And:* The result bit is 1 if both bits are 1; otherwise the result bit is 0.
  - *Not:* The result bit is 1 if the bit is 0; otherwise the result bit is 1.
  - *Or:* The result bit is 1 if either bit is 1; otherwise the result bit is 0.
  - *Xor:* The result bit is 1 if either bit is 1 but not both bits; otherwise the result bit is 0 (that is, 1 Xor 0 = 1, 1 Xor 1 = 0).

No overflows are possible from these operations. The enumerated type operators do the bitwise operation on the underlying type of the enumerated type, but the return value is the enumerated type.

Self Assessment

True or False:

7. Operator precedence determines the grouping of terms in an expression.
8. An action is a signal that informs an application that something important has occurred.
9. The mathematical operators compare values to one other.
10. The And, Not, Or, and Xor operators are called the mathematical operators.

2.3 Data Types

A data type is a class that is primarily used just to hold data. This is different from most other classes since they're primarily intended to 'do something', like access a database or format a page for example.

Table 2.5: Data Types

Data Type	# of Bytes	Values
Boolean	1	True or False
Byte	1	Unsigned
Char	2	Unicode character
Single	4	32-bit Floating Point Number
Double	8	64-bit Floating Point Number
Integer	4	32-bit Signed Integer

Contd...

Notes

Int16	2	16-bit Signed Integer
Int32	4	32-bit Signed Integer
Int64	8	64-bit Signed Integer
Long	8	64-bit Signed Integer
SByte	1	8-bit Signed Integer
Short	2	16-bit Signed Integer
UShort	2	16-bit Unsigned Integer
UInt16	2	16-bit Unsigned Integer
UInt32	4	32-bit Unsigned Integer
UInt64	8	64-bit Unsigned Integer
DateTime	8	Date and time of day
Decimal	16	Decimal number

Source: http://www.codeguru.com/vb/article.php/c18901/VBNET-Data-Types.htm

To specify the type for a variable we write,

```
Dim VariableName As DataType
```

Example:

```
Dim a As Boolean;
```

2.3.1 Comments on Data Types

Integer: *It* holds signed 32-bit (4-byte) integers ranging in value from -2,147,483,648 through 2,147,483,647. The Integer data type provides optimal performance on a 32-bit processor. The other integral types are slower to load and store from and to memory. The default value of Integer is 0.

Example:

```
Dim k As Integer
```

‘ The following statement causes an error because the value is too large.

```
k = 2147483648
```

‘ The following statement sets k to 6.

```
k = CInt(5.9)
```

Byte: It holds unsigned 8-bit (1-byte) integers ranging in value from 0 through 255. Use the Byte data type to contain binary data. The default value of Byte is 0.

String: Holds sequences of unsigned 16-bit (2-byte) code points ranging in value from 0 through 65535. Each *code point*, or character code, represents a single Unicode character. A string can contain from 0 to approximately 2 billion (2 ^ 31) Unicode characters. Use the String data type to hold multiple characters without the array management overhead of Char(), an array of Char elements. The default value of String is Nothing (a null reference). Note that this is not the same as the empty string (value “”).

Long: Holds signed 64-bit (8-byte) integers ranging in value from -9,223,372,036,854,775,808 through 9,223,372,036,854,775,807 (9.2...E+18). Use the Long data type to contain integer numbers that are too large to fit in the Integer data type. The default value of Long is 0.

Notes

Single: It holds signed IEEE 32-bit (4-byte) single-precision floating-point numbers ranging in value from -3.4028235E+38 through -1.401298E-45 for negative values and from 1.401298E-45 through 3.4028235E+38 for positive values. Single-precision numbers store an approximation of a real number.

Double: Holds signed IEEE 64-bit (8-byte) double-precision floating-point numbers ranging in value from -1.79769313486231570E+308 through -4.94065645841246544E-324 for negative values and from 4.94065645841246544E-324 through 1.79769313486231570E+308 for positive values. Double-precision numbers store an approximation of a real number. The Double data type provides the greatest and smallest possible magnitudes for a number. The default value of Double is 0.

Boolean: *It* holds values that can be only True or False. The keywords True and False correspond to the two states of Boolean variables. Use the Boolean data type to contain two-state values such as true/false, yes/no, or on/off. The default value of Boolean is False. When Visual Basic converts numeric data type values to Boolean, 0 becomes False and all other values become True. When Visual Basic converts Boolean values to numeric types, False becomes 0 and True becomes -1.

Caution Negative Numbers. Boolean is not a numeric type and cannot represent a negative value. In any case, you should not use Boolean to hold numeric values.

Example:

In the following example, runningVB is a Boolean variable, which stores a simple yes/no setting.

```
Dim runningVB As Boolean
' Check to see if program is running on Visual Basic engine.
If scriptEngine = "VB" Then
    runningVB = True
End If
```

Date: It holds IEEE 64-bit (8-byte) values that represent dates ranging from January 1 of the year 0001 through December 31 of the year 9999, and times from 12:00:00 AM (midnight) through 11:59:59.9999999 PM. Each increment represents 100 nanoseconds of elapsed time since the beginning of January 1 of the year 1 in the Gregorian calendar. The maximum value represents 100 nanoseconds before the beginning of January 1 of the year 10000.

Example:

Dim someDateAndTime As Date = #8/13/2002 12:14 PM#

Object: It holds 32-bit (4-byte) addresses that refer to objects. You can assign any reference type (string, array, class, or interface) to an Object variable. An Object variable can also refer to data of any value type (numeric, Boolean, Char, Date, structure, or enumeration).

Variant: The type of each element in a message is fixed and defined by the information in the component library. Because mainframe programs do not support the Variant data type, you must fix the type of each parameter at design time in Transaction Integrator (TI) Project. Microsoft Visual Basic Scripting Edition (VBScript), which is often used to create Active Server Pages (ASP) in Web-based applications, supports only the Variant data type. It does not accept declared variables. As a result, if your COM+ client application calls a TI Automation server and passes

Notes

parameters with Variant data types, the TI run-time environment forces each Variant data type into the type for each parameter as defined in the TI component library. The Variant data type is not supported in Visual Basic .NET. Visual Basic .NET supports defining data types as objects, and then casting the objects as data types. TI does not support variables defined as objects cast to data types. All method parameters must be defined initially as data types, not objects.

Example:

Variant named A can be explicitly declared as shown in either of these two examples:

```
Dim A
Dim A as Variant
```

Constants: They are declared using the keyword Const. Once declared, the value of these constants cannot be altered at run time.

```
Syntax:
 [Private | Public | Friend | Protected Friend ]
 Const constName As datatype = value
```

In the above syntax, the Public or Private can be used according to the scope of usage. The Value specifies the unchangeable value for the constant specified using the name constName.

Example:

```
Public Class Form1
   Public Const PI As Double = 3.14159
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
   Dim r, a As Single
   r = Val(TextBox1.Text)
   a = PI * r * r
   TextBox2.Text = a
   End Sub
   End Class
```

Case Study

Successful Migration of ERP-System with 650,000 Lines of Code from VB6 to .NET

Client Challenge

An Austrian manufacturing company with worldwide market presence relies on a highly optimized, in-house developed ERP-system based on Microsoft Basic Version 6 (VB6), consisting of 33 applications, covering almost all business processes.

Since April 2008 applications developed with the VB6 Integrated Development Environment (IDE) are no longer supported by Microsoft. In addition, the VB6.0 runtime will not be guaranteed after 2011. Therefore the following options were considered:

- Rewrite the system manually in Visual Basic .NET.
- Migrate the systems to .NET and improve the quality through refactoring.
- Find a package to replace the systems and undertake the required systems integration.

Contd...

Notes

The results of an extensive technical and commercial evaluation by SIS Datenverarbeitung GmbH showed a .NET migration to be the superior choice for the following reasons:

- Protection of extensive investment in highly optimized systems.
- Functional equivalence (1:1) of migrated applications.
- Short timeframe: full port within 12 months or less.
- Reduced risks and costs of migration vs. other options.

Chosen Approach

To facilitate a migration automated to the greatest possible extent SIS conducted an inventory of the existing code-base, identifying critical and "difficult" functions.

During the software evaluation phase SIS run a 25K prototype through all the conversion programs. It took 2.5 hours to get a compile-able and run-able VB.NET project with VB Migration Partner, and 13 hours with its closest competitor. The reduced effort, the ability to use programs and support for the convert-test-fix were the main reasons for choosing Code Architects' software.

After successful migration of the prototype the actual migration was performed as follows:

- Assurance of functional equivalence through detailed inventory of existing functions.
- Detailed migration-plan: tasks, restrictions, risk-analysis.
- Validation of tools, processes, results and cost.
- Migration infrastructure with multiple code-path: reference-, development-, test- and production-system.
- Configuration Management tool to track status of code.
- Extensive checklists for manual optimization during migration.
- Knowledge-base for future maintenance-guidelines.
- 5-step quality assurance:
    - 100%-code-Review, checks and refactoring of migrated code.
    - Unit-tests.
    - Function- and performance-tests.
    - Integration and trial-runs.
    - Performance profiling to validate performance after migration.

Benefits

The selected conversion software allowed to successfully migrating 650,000 lines of code within 6 months with a total effort of 18 man-months, excluding code review and refactoring. Another set of 300,000 lines of code will be migrated within three months.

Since the core applications of the ERP-System are mission critical to the client's operation, highest priority was assigned to achieve functional equivalence of the migrated systems: all functional changes were avoided to ensure that the old and new systems function identically.

VB Migration Partner's support library ensured that all VB-related methods and controls work identically in the converted VB.NET application, whereas its code generation engine prevented subtle behavioral differences from being accidentally introduced.

Contd...

Notes

Changes to "look and feel" and data-structures were also avoided to minimize education and training of users (approximately 350). Improved performance and software-quality was achieved through extensive refactoring.

Quote from the Customer

VB Migration Partner from Code Architects delivered fewer compilation and runtime errors than all its competitors. Its migration programs and the convert-test-fix methodology proved to be powerful and flexible enough to handle a large VB6 application (650K lines of code) with ease, the documentation is excellent, and Code Architects' tech support has always been very responsive.

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://www.vbmigration.com/casestudies/sis.aspx

Self Assessment

Fill in the blanks:

11. A ..... is a class that is primarily used just to hold data.
12. Boolean stores ..... and ..... .
13. ..... holds sequences of unsigned 16-bit (2-byte) code points ranging in value from 0 through 65535.
14. ..... holds 32-bit (4-byte) addresses that refer to objects.
15. ..... are declared using the keyword Const.

2.4 Summary

- Variables are used to temporarily store values during the execution of an application. Variables have a name and a type.
- Variables declared with the Dim statement within a procedure exist only as long as the procedure is executing.
- In VB. NET you don't have to declare a variable before using it.
- The scope of a variable defines which parts of your code are aware of its existence.
- Procedure-level variables are recognized only in the procedure in which they're declared.
- To make a module-level variable available to other modules, use the Public keyword to declare the variable.
- Variable of Type Object Holds 32-bit (4-byte) addresses that refer to objects. The Object data type is a reference type.
- The Application Object is used to refer to an application.
- The Database Object allows you to get a reference to the database you are using.
- Operator precedence determines the grouping of terms in an expression.

Notes

- An event is a signal that informs an application that something important has occurred.
- The relational operators compare values to one other.
- The And, Not, Or, and Xor operators are called the logical operators.
- A data type is a class that is primarily used just to hold data.

2.5 Keywords

Application Object: It is used to refer to an application.

Database Object: It allows you to get a reference to the database you are using.

Event: It is a signal that informs an application that something important has occurred.

Object: It holds 32-bit (4-byte) addresses that refer to objects.

Operator Precedence: It determines the grouping of terms in an expression.

Procedure-level Variables: They are recognized only in the procedure in which they're declared.

Scope: It defines which parts of your code are aware of its existence.

Variables: They are used to temporarily store values during the execution of an application.

2.6 Review Questions

1. Define a variable.
2. What do you mean by explicitly declaring a variable?
3. Explain scope of a variable.
4. Write a short note on variable lifetime.
5. Explain all built-in objects in VB.NET with examples.
6. Why do we need operators?
7. Is VB.NET event driven?
8. Explain the mathematical operators with suitable examples.
9. What do you mean my operator precedence?
10. List all major data types used in VB.NET.

Answers: Self Assessment

1.	Variables	2.	Name, type
3.	Dim	4.	Scope
5.	Procedure-level variables	6.	Module
7.	True	8.	False
9.	False	10.	False
11.	Data type	12.	True, False
13.	String	14.	Object
15.	Constants		

2.7 Further Readings

Notes

Books

Beginning Vb.Net 2003, Willis.

Building Distributed Applications with Visual Basic.NET, Dan Fox, Sams.

Object-oriented Programming with Visual Basic.Net, Hamilton.

Programming Visual Basic .NET, J. Liberty.

VB .NET Language in a Nutshell, Steven Roman, Ron Petrusha, Paul Lomax.

Visual Basic.NET Black Book, Steven Holzner.

Online links

http://vb.net-informations.com/language/vb.net_data_types.htm http://www.tutorialspoint.com/vb.net/vb.net_operators.htm

http://www.homeandlearn.co.uk/net/nets1p23.html http://www.jblearning.com/samples/0763724785/ch02_bronson.pdf

Notes

Unit 3: Decision Making and Looping

CONTENTS

Objectives

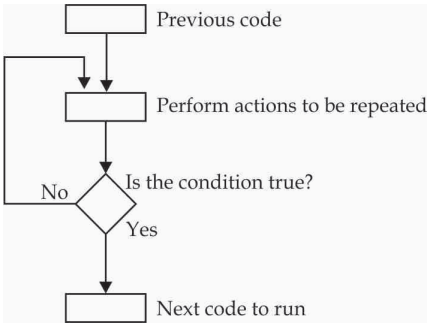
After studying this unit, you will be able to:

- Explain the concept of Looping in VB.NET
- Discuss IF statement and IF-Else statement
- Discuss While and Do-While statement
- Discuss For statement

Introduction

Visual Basic.NET loop structures allow you to run one or more lines of code repetitively. You can repeat the statements in a loop structure until a condition is True, until a condition is False, a specified number of times, or once for each element in a collection. The following illustration shows a loop structure that runs a set of statements until a condition becomes true. Running a set of statements until a condition becomes true.

Figure 3.1: Decision Structure



Source: http://msdn.microsoft.com/en-us/library/ezk76t25.aspx

Notes

3.1 Looping

A loop executes statements repeatedly. It often has an upper and lower bound. The For-loop proceeds through a range of values. A step indicates the progression. Other loops, such as While, continue until a condition is met. There may be a situation when you need to execute a block of code several number of times. In general statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths. Here are the looping constructs in the VB.NET language. The For Each construct is probably the least error-prone but is not always available. We also show examples of the For-loop construct directly on this page.

- For Each
- While
- Do While
- Do Until

Self Assessment

Fill in the blanks:

1. Visual Basic.NET ..... allow you to run one or more lines of code repetitively.
2. You can repeat the statements in a ..... structure.

3.2 If Statement

It conditionally executes a group of statements, depending on the value of an expression. Its syntax is,

```
If condition [ Then ]
    [ statements ]
[ ElseIf elseifcondition [ Then ]
    [ elseifstatements ] ]
[ Else
    [ elsestatements ] ]
End If
```

-or-

```
If condition Then [ statements ] [ Else [ elsestatements ] ]
```

where,

condition: Required. Expression. Must evaluate to True or False, or to a data type that is implicitly convertible to Boolean.

Then: Required in the single-line form, optional in the multiple-line form.

Statements : Optional. One or more statements following If...Then that are executed if condition evaluates to True.

Elseifcondition: Required if ElseIf is present. Expression. Must evaluate to True or False, or to a data type that is implicitly convertible to Boolean.

Notes

elseifstatements: Optional. One or more statements following ElseIf...Then that are executed if elseifcondition evaluates to True.

elsestatements: Optional. One or more statements that are executed if no previous condition or elseifcondition expression evaluates to True.

End If: Terminates the If...Then...Else block.

It is possible to have multiple statements executed as the result of an If...Then decision. All statements must be on the same line and be separated by colons.

Example:

If A > 10 Then A = A + 1 : B = B + A : C = C + B

In the multiple-line form, the If statement must be the only statement on the first line.

Example:

```
Dim num1 As Integer
Dim num2 As Integer
num1 = 39
num2 = 40
If num1 > num2 Then
MsgBox ("num1 is greater")
Else
MsgBox ("num2 is greater")
End If
```

Self Assessment

True or False:

3. Condition in an If statement is an optional one.
4. Then is required in the multi-line form of an If statement.
5. End If terminates the If...Then...Else block.

3.3 If-Else Statement

You can use the single-line form for short, simple tests. However, the multiple-line form provides more structure and flexibility than the single-line form and is usually easier to read, maintain, and debug. When a multiple-line If...Then...Else is encountered, condition is tested. If condition is True, the statements following Then are executed. If condition is False, each ElseIf statement is evaluated in order. When a True elseifcondition is found, the statements immediately following the associated Then are executed. If no elseifcondition evaluates to True, or if there are no ElseIf statements, the statements following Else are executed. After executing the statements following Then, ElseIf, or Else, execution continues with the statement following End If. In the multiple-line form, the If statement must be the only statement on the first line. The ElseIf, Else, and End If statements can be preceded only by a line label. The multiple-line If...Then...Else block must end with an End If statement.

Notes

To determine whether or not an If statement introduces a multiple-line form, examine what follows the Then keyword. If anything other than a comment appears after Then in the same statement, the statement is treated as a single-line If statement. If Then is absent, it must be the beginning of a multiple-line If...Then...Else. The ElseIf and Else clauses are both optional. You can have as many ElseIf clauses as you want in a multiple-line If...Then...Else, but none can appear after an Else clause. Multiple-line forms can be nested within one another.

Example:

```
Dim number, digits As Integer
Dim myString As String
number = 53
If number < 10 Then
    digits = 1
ElseIf number < 100 Then
    digits = 2
Else
    digits = 3
End If
If digits = 1 Then myString = "One" Else myString = "More than one"
```

In the preceding example, the ElseIf condition evaluates to True, and digits is assigned a value of 2. The last statement then assigns a value of "More than one" to myString.

3.3.1 Select Case

It runs one of several groups of statements, depending on the value of an expression. Its syntax is,

```
Select [ Case ] testexpression
    [ Case expressionlist
    [ statements ] ]
    [ Case Else
    [ elsestatements ] ]
End Select
```

where,

testexpression: Required. Expression. Must evaluate to one of the elementary data types (Boolean, Byte, Char, Date, Double, Decimal, Integer, Long, Object, SByte, Short, Single, String, UInteger, ULong, and UShort).

Expressionlist: Required in a Case statement. List of expression clauses representing match values for testexpression. Multiple expression clauses are separated by commas. Each clause can take one of the following forms:

- expression1 To expression2
- [Is] comparison operator expression
- expression

Notes

Use the To keyword to specify the boundaries of a range of match values for testexpression. The value of expression1 must be less than or equal to the value of expression2.

Use the Is keyword with a comparison operator (=, <>, <, <=, >, or >=) to specify a restriction on the match values for testexpression. If the Is keyword is not supplied, it is automatically inserted before comparison operator.

The form specifying only expression is treated as a special case of the Is form where comparison operator is the equal sign (=). This form is evaluated as testexpression = expression.

The expressions in expressionlist can be of any data type, provided they are implicitly convertible to the type of testexpression and the appropriate comparison operator is valid for the two types it is being used with.

Statements: Optional. One or more statements following Case that run if testexpression matches any clause in expressionlist.

elsestatements : Optional. One or more statements following Case Else that run if testexpression does not match any clause in the expressionlist of any of the Case statements.

End Select : Terminates the definition of the Select...Case construction.

If testexpression matches any Case expressionlist clause, the statements following that Case statement run up to the next Case, Case Else, or End Select statement. Control then passes to the statement following End Select. If testexpression matches an expressionlist clause in more than one Case clause, only the statements following the first match run. The Case Else statement is used to introduce the elsestatements to run if no match is found between the testexpression and an expressionlist clause in any of the other Case statements. Although not required, it is a good idea to have a Case Else statement in your Select Case construction to handle unforeseen testexpression values. If no Case expressionlist clause matches testexpression and there is no Case Else statement, control passes to the statement following End Select. You can use multiple expressions or ranges in each Case clause. For example, the following line is valid.

```
Case 1 To 4, 7 To 9, 11, 13, Is > maxNumber
```

Note The Is keyword used in the Case and Case Else statements is not the same as the Is Operator (Visual Basic), which is used for object reference comparison.

You can specify ranges and multiple expressions for character strings. In the following example, Case matches any string that is exactly equal to “apples”, has a value between “nuts” and “soup” in alphabetical order, or contains the exact same value as the current value of testItem.

```
Case “apples”, “nuts” To “soup”, testItem
```

The setting of Option Compare can affect string comparisons. Under Option Compare Text, the strings “Apples” and “apples” compare as equal, but under Option Compare Binary, they do not.

Note A Case statement with multiple clauses can exhibit behavior known as short-circuiting. Visual Basic.NET evaluates the clauses from left to right, and if one produces a match with testexpression, the remaining clauses are not evaluated. Short-circuiting can improve performance, but it can produce unexpected results if you are expecting every expression in expressionlist to be evaluated.

Notes

If the code within a Case or Case Else statement block does not need to run any more of the statements in the block, it can exit the block by using the Exit Select statement. This transfers control immediately to the statement following End Select.

Select Case constructions can be nested. Each nested Select Case construction must have a matching End Select statement and must be completely contained within a single Case or Case Else statement block of the outer Select Case construction within which it is nested.

```
Example:

Select Case agerange

Case 16 To 21
MsgBox "Still Young"
Case 50 To 64
MsgBox "Start Lying"

End Select
```

Self Assessment

True or False:

6. Select Case runs one of several groups of statements, depending on the value of an expression.
7. You cannot specify ranges and multiple expressions for character strings in Select Case.

3.4 While Statement

It executes a series of statements as long as a given condition is True. The syntax for this loop construct is:

```
While condition
    [ statements ]
    [ Continue While ]
    [ statements ]
    [ Exit While ]
    [ statements ]
End While
```

Here statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is logical true. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop. Here key point of the *While* loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

Example:

```
Sub Main()
Dim a As Integer = 10
' while loop execution '
While a < 20
Console.WriteLine("value of a: {0}", a)
a = a + 1
End While
```

Notes

```
  Console.ReadLine()
  End Sub
Output:
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

```
Exit While
```

The Exit statement transfers the control from a procedure or block immediately to the statement following the procedure call or the block definition. It terminates the loop, procedure, try block or the select block from where it is called. If you are using nested loops (i.e., one loop inside another loop), the Exit statement will stop the execution of the innermost loop and start executing the next line of code after the block.

The syntax for the Exit statement is:

```
Exit { While }
```

Example:

```
Sub Main()
    ' local variable definition
    Dim a As Integer = 10
    ' while loop execution '
    While (a < 20)
        Console.WriteLine("value of a: {0}", a)
        a = a + 1
    End While
    Console.ReadLine()
End Sub
Output:
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
```

Self Assessment

Fill in the blanks:

8. When the condition becomes ....., program control passes to the line immediately.
9. The ..... statement transfers the control from a procedure or block immediately to the statement following the procedure call or the block definition.
10. If you are using nested loops the Exit statement will stop the execution of the ..... loop.

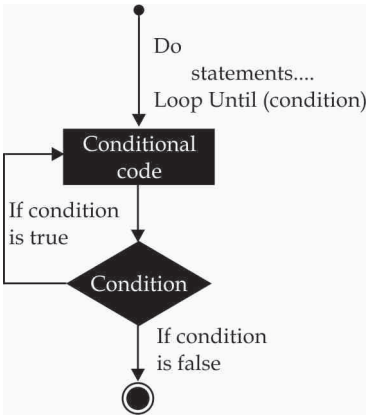
3.5 Do-While Statement

Notes

It repeats the enclosed block of statements while a Boolean condition is True or until the condition becomes True. It could be terminated at any time with the Exit Do statement. The syntax for this loop construct is:

```
Do
    [ statements ]
    [ Continue Do ]
    [ statements ]
    [ Exit Do ]
    [ statements ]
Loop { While | Until } condition
```

Figure 3.2: Do Loop



Source: http://www.tutorialspoint.com/vb.net/vb.net_do_loops.htm

Example:

```
Sub Main()
    ' local variable definition
    Dim a As Integer = 10
    'do loop execution
    Do
        Console.WriteLine("value of a: {0}", a)
        a = a + 1
    Loop While (a < 20)
    Console.ReadLine()
End Sub
Output:
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Notes

3.5.1 Do Until Loop

Do Loop Until Statement executes a set of statements until a condition becomes false, this is an infinite loop might have to terminated using Ctrl + Break. Its syntax is,

```
Do
    [Statements]
Loop Until [Condition]
```

In the above syntax the Statements are executed until the Condition becomes false.

Example:

```
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    Dim X As String
    Do
        X$ = InputBox$("Correct Password Please")
    Loop Until X$ = "Ranger"
    End Sub
```

In the above Do Until Loop example, a input box is displayed until the correct password is typed.

Example:

Self Assessment

True or False:

11. Do-While Statement repeats the enclosed block of statements while a Boolean condition is False
12. Do-While Statement could be terminated at any time with the Exit Do statement

3.6 For Statement

For next loop statement executes a set of statements repeatedly in a loop for the given initial, final value range with the specified step by step increment or decrement value.

Its syntax is,

```
For counter = start To end [Step]
    [Statement]
Next [counter]
```

In the above syntax the Counter is range of values specified using the Start, End parameters. The Step specifies step increment or decrement value of the counter for which the statements are executed.

Example:

```
Private Sub Form1_Load(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles MyBase.Load
    Dim i As Integer
    Dim j As Integer
```

Notes

```
    j = 0
    For i = 1 To 10 Step 1
        j = j + 1
        MsgBox ("Value of j is::" & j)
    Next i
End Sub
```

In the above For Next Loop example the counter value of i is set to be in the range of 1 to 10 and is incremented by 1. The value of j is increased by 1 for 10 times as the loop is repeated.

Step Argument

The Step keyword is used near the end of the For-statement and it is the delta each loop iteration will have. If you want to decrement by n each time, you can use -n. If you want to increment by n, use n.

Counter Argument

In executing a For. . .Next loop, Visual Basic does the following:

1. Sets counter equal to start.
2. Tests to see whether counter is greater than end. If so, it exits the loop without executing the statements in the loop's body, not even once. If increment is negative, Visual Basic tests to see whether counter is less than end. If it is, it exits the loop.
3. Executes the statements in the block.
4. Increases counter by the amount specified with the increment argument, following the Step keyword. If the increment argument isn't specified, counter is increased by 1. If Step is a negative value, counter is decreased accordingly.
5. Continues with step 3.

Nesting Loops

VB.NET allows using one loop inside another loop. Following section shows few examples to illustrate the concept. The syntax for a nested For loop statement in VB.NET is as follows:

```
For counter1 [ As datatype1 ] = start1 To end1 [ Step step1 ]
    For counter2 [ As datatype2 ] = start2 To end2 [ Step step2 ]
    ...
    Next [ counter2 ]
Next [ counter 1]
```

Example:

```
 Dim i, j As Integer // local variable definition

 For i = 2 To 100
     For j = 2 To i
                If ((i Mod j) = 0) Then
             Exit For
        End If
    Next j
    If (j > (i \ j)) Then
```

Notes

```
        Console.WriteLine("{0} is prime", i)
    End If
Next i
Console.ReadLine()
Output:
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
```

Exit For

We can exit a loop explicitly using the exit statement.

Example:

```
Dim valExit = 0
For Each
    For Each
        If condition Then
            valExit = 1
            Exit For
        End If
    Next

    If valExit = 1 Then
        Exit For
    End If
Next
```

Notes

Case Study

.NET History

The Microsoft .NET is a new internet technology or rather strategy introduced by Microsoft. .NET was originally known as the NGWS (Next Generation Windows Services) which was said to be an Internet based platform of Next Generation Windows Services. Before the official announcement of .NET, NGWS was the term used to describe the above phrase.

The .NET Framework was first released in beta in November 2000 and development continues today. Each version of the framework has betas, final versions, service packs, and patches associated with it.

Table 1

Version	Version Number	Release Date	Visual Studio	Default in Windows
1.0	1.0.3705.0	02/13/2002	VisualStudio.NET	
1.1	1.1.4322.573	04/24/2003	Visual Studio .NET 2003	Windows Server 2003
2.0	2.0.50727.42	11/07/2005	Visual Studio 2005	
3.0	3.0.4506.30	11/06/2006		Windows Vista, Windows Server 2008
3.5	3.5.21022.8	11/19/2007	Visual Studio 2008	Windows 7, Windows Server 2008 R2
4.0	4.0.30319.1	04/12/2010	Visual Studio 2010	

.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.

Contd...

Notes

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 non-privileged 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 application performs. 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 application begins to execute.

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://dotnet.etisbew.com/dotnet-history.html

Self Assessment

Notes

Fill in the blanks:

13. VB.NET allows using one loop inside another loop and this is called ..... of loops.
14. For ..... loop statement executes a set of statements repeatedly in a loop.
15. We can exit a loop explicitly using the ..... statement.

3.7 Summary

- Visual Basic.NET loop structures allow you to run one or more lines of code repetitively.
- You can repeat the statements in a loop structure until a condition is True, until a condition is False, a specified number of times, or once for each element in a collection.
- Select Case runs one of several groups of statements, depending on the value of an expression.
- While Statement executes a series of statements as long as a given condition is True.
- The Exit statement transfers the control from a procedure or block immediately to the statement following the procedure call or the block definition.
- If you are using nested loops (i.e., one loop inside another loop), the Exit statement will stop the execution of the innermost loop and start executing the next line of code after the block.
- Do-While Statement repeats the enclosed block of statements while a Boolean condition is True.
- Do Loop Until Statement executes a set of statements until a condition becomes false.
- For next loop statement executes a set of statements repeatedly in a loop.
- The Step keyword is used near the end of the For-statement and it is the delta each loop iteration will have.
- VB.NET allows using one loop inside another loop.

3.8 Keywords

Do Loop Until Statement: It executes a set of statements until a condition becomes false.

Do-While Statement: It repeats the enclosed block of statements while a Boolean condition is True.

Exit statement: It transfers the control from a procedure or block immediately to the statement following the procedure call or the block definition.

For next loop statement: It executes a set of statements repeatedly in a loop.

Loop structures: They allow you to run one or more lines of code repetitively.

Nesting: VB.NET allows using one loop inside another loop.

Select Case: It runs one of several groups of statements, depending on the value of an expression.

Step keyword: It is used near the end of the For-statement and it is the delta each loop iteration will have.

While Statement: It executes a series of statements as long as a given condition is True.

Notes

3.9 Review Questions

1. What is a loop?
2. Why do we need loops?
3. Write a note on if statement with an example.
4. Compare if and if-elseif constructs.
5. Explain the working of while loop.
6. Differentiate between while and do while loops.
7. Why do we use the exit keyword?
8. What is the syntax of the for next loop?
9. Why is a counter used in the for next loop?
10. What do you mean by nesting of loops?

Answers: Self Assessment

1.	Loop structures	2.	Loop
3.	False	4.	False
5.	True	6.	True
7.	False	8.	False
9.	Exit	10.	Innermost
11.	False	12.	True
13.	Nesting	14.	Next
15.	Exit		

3.10 Further Readings

Books

Beginning Vb.Net 2003, Willis.

Object-oriented Programming with Visual Basic.Net, Hamilton.

Programming Visual Basic .NET, J. Liberty.

Visual Basic.NET Black Book, Steven Holzner.

Online links

http://msdn.microsoft.com/en-US/library/zh1f56zs%28v=vs.80%29.aspx

http://www.tutorialspoint.com/vb.net/vb.net_loops.htm

http://www.informit.com/library/content.aspx?b=Net_2003_21days&seqNum=97

http://www.jblearning.com/samples/0763724785/ch02_bronson.pdf

Notes

Unit 4: Array

CONTENTS

Objectives

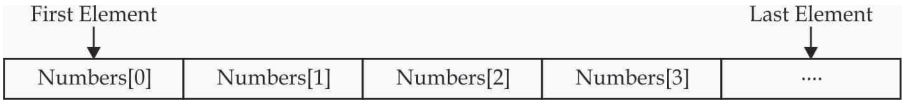
After studying this unit, you will be able to:

- Understand arrays and its types
- Define the split function
- Discuss the join function
- Explain multidimensional arrays
- Discuss system. array class

Introduction

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

Figure 4.1: Array



Source: http://www.tutorialspoint.com/vb.net/vb.net_arrays.htm

Notes

To declare an array in VB.NET, you use the Dim statement.

Example:

```
Dim intData(30)                    // an array of 31 elements
Dim strData(20) As String          // an array of 21 strings
Dim twoDarray(10, 20) As Integer   // a two dimensional array of integers
Dim ranges(10, 100)                // a two dimensional array
```

You can also initialize the array elements while declaring the array.

Example:

```
Dim intData() As Integer = {12, 16, 20, 24, 28, 32}
Dim names() As String = {"Karthik", "Sandhya", _
"Shivangi", "Ashwitha", "Somnath"}
Dim miscData() As Object = {"Hello World", 12d, 16ui, "A"c}
```

The elements in an array can be stored and accessed by using the index of the array.

Example:

```
Sub Main()
   Dim n(10) As Integer    // n is an array of 11 integers
   Dim i, j As Integer

   For i = 0 To 10         // initialize elements of array n
      n(i) = i + 100       // set element at location i to i + 100
   Next i
      For j = 0 To 10      // output each array element's value
     Console.WriteLine("Element({0}) = {1}", j, n(j))
   Next j
   Console.ReadKey()
End Sub
```

```
Output:

Element(0) = 100
Element(1) = 101
Element(2) = 102
Element(3) = 103
Element(4) = 104
Element(5) = 105
Element(6) = 106
Element(7) = 107
Element(8) = 108
Element(9) = 109
Element(10) = 110
```

4.1 Types of Arrays

There are two major types of arrays. Let us discuss them in detail.

Notes

4.1.1 Fixed-Size Arrays

An array is declared by simply using parenthesis to indicate that a variable has an index. For example, Dim StringArray() is syntactically correct (although at least the size of the array will eventually have to be supplied).

A collection is typically an instance of a class and has to be declared with the New keyword to create the instance. Elements are added and deleted using methods. This code shows the difference.

```
Dim StringArray()
ReDim StringArray(10)
StringArray(0) = "ABCDEF"

Dim ListCollection As New List(Of String)
ListCollection.Add("ABCDEF")
ListCollection.Remove("ABCDEF")
```

4.1.2 Dynamic Arrays

Dynamic arrays are arrays that can be dimensioned and re-dimensioned as par the need of the program. You can declare a dynamic array using the ReDim statement.

```
Syntax for ReDim statement:
ReDim [Preserve] arrayname(subscripts)
```

Where,

- The Preserve keyword helps to preserve the data in an existing array, when you resize it.
- arrayname is the name of the array to re-dimension
- subscripts specifies the new dimension.

Example:

```
Sub Main()
    Dim marks() As Integer
    ReDim marks(2)
    marks(0) = 85
    marks(1) = 75
    marks(2) = 90
    ReDim Preserve marks(10)
    marks(3) = 80
    marks(4) = 76
    marks(5) = 92
    marks(6) = 99
    marks(7) = 79
    marks(8) = 75
    For i = 0 To 10
        Console.WriteLine(i & vbTab & marks(i))
    Next i
    Console.ReadKey()
End Sub
Output:
0   85
1   75
```

Notes

```
2   90
3   80
4   76
5   92
6   99
7   79
8   75
9   0
10  0
```

Self Assessment

Fill in the blanks:

1. An ..... stores a fixed-size sequential collection of elements of the same type.
2. All arrays consist of ..... memory locations.
3. To declare an array in VB.Net, you use the ..... statement.
4. A ..... is typically an instance of a class and has to be declared with the New keyword to create the instance.

4.2 Split Function

The string split function returns array of strings which are the splits of the given string it is delimited by the given System.Char array

Public Function Split (ByVal ParamArray separator () As Char) As String ()

Parameters:

Separator - the given delimiter

Returns:

An array of Strings delimited by one or more characters in separator

```
Public Class Form1
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click

Dim str As String
Dim strArr() As String
Dim count As Integer
str = "God is great"
strArr = str.Split(" ")
For count = 0 To strArr.Length - 1
MsgBox(strArr(count))
Next

End Sub
End Class
```

When you run this code, you will get "God" "is" "great" in separate messagebox.

Notes

Self Assessment

True or False:

5. The string split function returns a single string.
6. The split function output is delimited by one or more characters in separator.

4.3 Join Function

The Join method is used when you want to join the elements of an array back together again.

Example:

```
Dim LineOfText As String
Dim i As Integer
Dim aryTextFile(3) As String

aryTextFile(0) = "UserName1"
aryTextFile(1) = "Password1"
aryTextFile(2) = "UserName2"
aryTextFile(3) = "Password2"

LineOfText = String.Join( "-", aryTextFile )

MsgBox( LineOfText )
```

The line that joins each element in the array is this:

```
LineOfText = String.Join( "-", aryTextFile )
```

Note If you have an older version of the VB.NET software, use LineOfText.Join instead of String.Join.

In between the round brackets of Join(), you first type what you want to use as a separator. Here, we're using an hyphen as a separator. Next, you put the name of your array. Again the round brackets from the array have gone missing.

When the line executes, the variable LineOfText will hold the following:

```
"UserName1-Password1-UserName2-Password2"
```

Once you have the array elements joined together, you could then write the line back to your text file. Split and Join can be very useful indeed. Especially when you're working with text files.

Self Assessment

True or False:

7. The split method is used when you want to join the elements of an array back together again.
8. Once you have the array elements joined together, you could then write the line back to your text file.

Notes

4.4 Adding New Elements

To add new elements in an array and preserving the old values use ReDim with Preserve keyword. ReDim in loop is advisable when you have no idea about the size and came to know for increasing the Array size one by one.

Example:

```
Dim TeamIndex(), i As Integer

For i = 0 to 100
 ReDim Preserve TeamIndex(i)
 TeamIndex(i) = <some value>
Next
If you to declare the size of array at later in code in shot then use
 ReDim TeamIndex(100)
So the code will be :
Dim TeamIndex(), i As Integer
ReDim TeamIndex(100)
For i = 0 to 100
 TeamIndex(i) = <some value>
Next
You can Use the ArrayList to use Add/Remove the values more dynamically.
 Sub Main()
 Dim list As New ArrayList // Create an ArrayList and add three strings to
it
 list.Add("Dot")
 list.Add("Net")
 list.Add("Perls")
 list.RemoveAt(1) // Remove a string.
 ' Insert a string.
 list.Insert(0, "Carrot")
 list.RemoveRange(0, 2)  // Remove a range.
 Dim str As String  // Remove a range
 For Each str In list
   Console.WriteLine(str)
 Next
 End Sub
```

4.4.1 Erasing an Array

Erase statement is used to release array variables and deallocate the memory used for their elements. Its syntax is,

Erase arraylist

Where arraylist is a required parameter It is a list of array variables to be erased. It consists of multiple variables are separated by commas.

Notes

The Erase statement can appear only at procedure level. This means you can release arrays inside a procedure but not at class or module level. The Erase statement is equivalent to assigning Nothing to each array variable.

Example:

The following example uses the Erase statement to clear two arrays and free their memory (1000 and 100 storage elements, respectively). The ReDim statement then assigns a new array instance to the three-dimensional array.

```
Dim threeDimArray(9, 9), twoDimArray(9, 9) As Integer
Erase threeDimArray, twoDimArray
ReDim threeDimArray(4, 4, 9)
```

Self Assessment

Fill in the blanks:

9. To add new elements in an array use ..... statement.
10. To preserve the old values use ..... keyword with Redim statement.
11. ..... statement is used to release array variables and deallocate the memory used for their elements.

4.5 Multidimensional Arrays

One-dimensional arrays, such as those presented so far, are good for storing long sequences of one-dimensional data (such as names or temperatures). But how would you store a list of cities and their average temperatures in an array? Or names and scores; years and profits; or data with more than two dimensions, such as products, prices, and units in stock? In some situations, you will want to store sequences of multidimensional data. You can store the same data more conveniently in an array of as many dimensions as needed.

Figure 4.2 shows two one-dimensional arrays — one of them with city names, the other with temperatures. The name of the third city would be City(2), and its temperature would be Temperature(2).

Figure 4.2: Two One-dimensional Arrays and the Equivalent Two-dimensional Array

	Cities (7)	Temperatures (7)	Temperatures (7, 1)	
0	San Francisco	78	San Francisco	78
1	Los Angeles	86	Los Angeles	86
2				
3				
4				
5				
6				
7	Seattle	65	Seattle	65
	Two one-dimensional arrays		A two-dimensional array	

Source: http://visualbasic.w3computing.com/vb2008/2/vb-multidimensional-arrays.php

A two-dimensional array has two indices: The first identifies the row (the order of the city in the array), and the second identifies the column (city or temperature). To access the name and temperature of the third city in the two-dimensional array, use the following indices:

Notes

Example:

Temperatures(2, 0) ' is the third city's name

Temperatures(2, 1) ' is the third city's average temperature

The benefit of using multidimensional arrays is that they're conceptually easier to manage. Suppose that you're writing a game and want to track the positions of certain pieces on a board. Each square on the board is identified by two numbers: its horizontal and vertical coordinates. The obvious structure for tracking the board's squares is a two-dimensional array, in which the first index corresponds to the row number, and the second corresponds to the column number.

Example:

Dim Board(9, 9) As Integer

When a piece is moved from the square in the first row and first column to the square in the third row and fifth column, you assign the value 0 to the element that corresponds to the initial position:

Board(0, 0) = 0

And you assign 1 to the square to which it was moved to indicate the new state of the board:

Board(2, 4) = 1

To find out whether a piece is on the top-left square, you'd use the following statement:

Example:

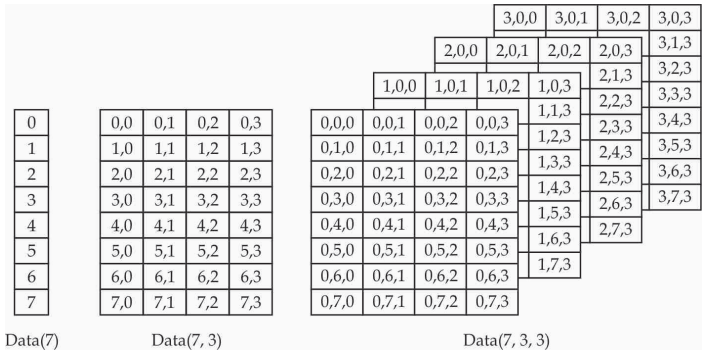
```
If Board(0, 0) = 1 Then
{ piece found}
Else
{ empty square}
End If
```

This notation can be extended to more than two dimensions. The following statement creates an array with 1,000 elements (10 by 10 by 10):

Dim Matrix(9, 9, 9)

You can think of a three-dimensional array as a cube made up of overlaid two-dimensional arrays, such as the one shown in Figure 4.3.

Figure 4.3: Pictorial Representations of One-, Two-, and Three-dimensional Arrays



Source: http://visualbasic.w3computing.com/vb2008/2/vb-multidimensional-arrays.php

Notes

It is possible to initialize a multidimensional array with a single statement, just as you do with a one-dimensional array. You must insert enough commas in the parentheses following the array name to indicate the array's rank. The following statements initialize a two-dimensional array and then print a couple of its elements:

Example:

```
Dim a(,) As Integer = {{10, 20, 30}, {11, 21, 31}, {12, 22, 32}}
Console.WriteLine(a(0, 1)) ' will print 20
Console.WriteLine(a(2, 2)) ' will print 32
```

You should break the line that initializes the dimensions of the array into multiple lines to make your code easier to read. Just insert the line continuation character at the end of each continued line:

Example:

```
Dim a(,) As Integer = {{10, 20, 30},
{11, 21, 31},
{12, 22, 32}}
```

If the array has more than one dimension, you can find out the number of dimensions with the Array.Rank property. Let's say you have declared an array for storing names and salaries by using the following statements:

Dim Employees(1,99) As Employee

To find out the number of dimensions, use the following statement:

Employees.Rank

When using the Length property to find out the number of elements in a multidimensional array, you will get back the total number of elements in the array (2 × 100 for our example). To find out the number of elements in a specific dimension, use the GetLength method, passing as an argument a specific dimension. The following expressions will return the number of elements in the two dimensions of the array:

Example:

```
Debug.WriteLine(Employees.GetLength(0))
2
Debug.WriteLine(Employees.GetLength(1))
100
```

Because the index of the first array element is zero, the index of the last element is the length of the array minus 1. Let's say you have declared an array with the following statement to store player statistics for 15 players, and there are five values per player:

Dim Statistics(14, 4) As Integer

Example:

The following statements will return the highlighted values shown beneath them:

```
Debug.WriteLine(Statistics.Rank)
2                        // dimensions in array
Debug.WriteLine(Statistics.Length)
```

Notes

```
75                      // total elements in array
Debug.WriteLine(Statistics.GetLength(0))
15                      // elements in first dimension
Debug.WriteLine(Statistics.GetLength(1))
5                        // elements in second dimension
Debug.WriteLine(Statistics.GetUpperBound(0))
14                       // last index in the first dimension
Debug.WriteLine(Statistics.GetUpperBound(1))
4                        // last index in the second dimension
```

Self Assessment

Fill in the blanks:

12. A two-dimensional array has ..... indices.
13. When using the ..... property to find out the number of elements in a multidimensional array, you will get back the total number of elements in the array.

4.6 System. Array Class

Provides methods for creating, manipulating, searching, and sorting arrays, thereby serving as the base class for all arrays in the common language runtime.

Example:

```
'Declaration
<SerializableAttribute> _
<ComVisibleAttribute(True)> _
Public MustInherit Class Array _
   Implements ICloneable, IList, ICollection, IEnumerable, _
   IStructuralComparable, IStructuralEquatable
```

Table 4.1: Properties Array Class

Name	Description
IsFixedSize	Gets a value indicating whether the Array has a fixed size.
IsReadOnly	Get a value indicating whether the Array is read-only.
IsSynchronized	Gets a value indicating whether across to the Array is Synchronized (thread safe).
Length	Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
LongLength	Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
Rank	Gets the rank (number of dimensions) of the Array. For example, a one-dimensional array returns 1, a two-dimensional array returns 2, and so on.
SyncRoot	Gets an object that can be used to synchronize access to the Array.

Source: http://msdn.microsoft.com/en-us/library/system.array.aspx

Notes

Let us discuss one of the methods of the System.Array class.

```
Array.CreateInstance Method (Type, Int32)
Creates a one-dimensional Array of the specified Type and length, with
zero-based indexing.

Syntax:
Public Shared Function CreateInstance ( elementType As Type, length As
Integer ) As Array
Parameters:

elementType - The Type of the Array to create.
Length - The size of the Array to create.
```

Example:

The following code example shows how to create and initialize a one-dimensional Array.

```
Imports System
Public Class SamplesArray

 Public Shared Sub Main()

 //Creates and initializes a one-dimensional Array of type Int32.
 Dim my1DArray As Array = Array.CreateInstance(GetType(Int32), 5)
 Dim i As Integer
 For i = my1DArray.GetLowerBound(0) To my1DArray.GetUpperBound(0)
 my1DArray.SetValue(i + 1, i)
 Next i
 // Displays the values of the Array.
 Console.WriteLine("The one-dimensional Array contains the " _
 + "following values:")
 PrintValues(my1DArray)

 End Sub

 Public Shared Sub PrintValues(myArr As Array)
 Dim myEnumerator As System.Collections.IEnumerator = _
 myArr.GetEnumerator()
 Dim i As Integer = 0
 Dim cols As Integer = myArr.GetLength((myArr.Rank - 1))
 While myEnumerator.MoveNext()
 If i < cols Then
 i += 1
 Else
 Console.WriteLine()
 i = 1
 End If
 Console.Write(ControlChars.Tab + "{0}", myEnumerator.Current)
 End While
 Console.WriteLine()
 End Sub
End Class
```

Notes

```
Output:
The one-dimensional Array contains the following values:
1 2 3 4 5
```

4.7 Retrieving the Contents of an Array

Accessing an element in an array is similar to assigning a value to an array element. All that is required is the array name and the index of the element to be accessed. The following code excerpt displays MessageBoxes containing the string value the first and seconds elements in the strColors array:

```
Dim strColors(5) As String = {"Red", "Green", "Blue", "Indigo", "Violet",
"Yellow" }

MessageBox.Show(strColors(0))
MessageBox.Show(strColors(1))
```

A *For* loop can also be used to iterate through each element of an array:

Example:

```
Dim strColors(5) As String = {"Red", "Green", "Blue", "Indigo", "Violet",
"Yellow" }
Dim intCount As Integer

For intCount = 0 To 10
  MessageBox.Show(strColors(intCount))
Next intCount
```

When you use a For...Next loop to iterate through an array, you may be inclined to hard code the starting and ending counter values. For instance, suppose you created the following array:

Example:

```
MusicGenres = Array("Blues", "Classic Rock", "Country",
"Dance", "Disco", "Funk", "Grunge", "Hip -Hop",
"Jazz", "Metal", "New Age", "Oldies", "Other")
```

you might think to loop through the array like so

```
For x = 0 To 12
  Debug.Print MusicGenres(x)
Next x
```

However, because you may want to add more items to the array at a later time, it's best to use the LBound() and UBound() functions to delimit the counter's boundaries, as in

```
For x = LBound(MusicGenres) To UBound(MusicGenres)
Debug.Print MusicGenres(x)
Next x
```

This way, no matter how many times you add items to the array, you won't need to modify the For...Next loop at all.

Notes

Note You don't need to subtract 1 from the UBound() value because the function returns the array's largest available subscript NOT the number of items in the array.

Case Study

Array Class

Pointer-based arrays have a number of problems. For example, a program can easily "walk off" either end of an array, because C++ does not check whether subscripts fall outside the range of an array (the programmer can still do this explicitly though). Arrays of size n must number their elements 0, ..., n 1; alternate subscript ranges are not allowed. An entire non-char array cannot be input or output at once; each array element must be read or written individually. Two arrays cannot be meaningfully compared with equality operators or relational operators (because the array names are simply pointers to where the arrays begin in memory and, of course, two arrays will always be at different memory locations). When an array is passed to a general-purpose function designed to handle arrays of any size, the size of the array must be passed as an additional argument. One array cannot be assigned to another with the assignment operator(s) (because array names are constant pointers and a constant pointer cannot be used on the left side of an assignment operator). These and other capabilities certainly seem like "naturals" for dealing with arrays, but pointer-based arrays do not provide such capabilities. However, C++ does provide the means to implement such array capabilities through the use of classes and operator overloading.

In this example, we create a powerful array class that performs range checking to ensure that subscripts remain within the bounds of the Array. The class allows one array object to be assigned to another with the assignment operator. Objects of the Array class know their size, so the size does not need to be passed separately as an argument when passing an Array to a function. Entire Arrays can be input or output with the stream extraction and stream insertion operators, respectively. Array comparisons can be made with the equality operators == and !=.

This example will sharpen your appreciation of data abstraction. You will probably want to suggest other enhancements to this Array class. Class development is an interesting, creative and intellectually challenging activity, always with the goal of "crafting valuable classes."

```
Array.h
// Array class for storing arrays of integers.
#ifndef ARRAY_H
#define ARRAY_H

#include <iostream>
using std::ostream;
using std::istream;

```

Contd...

Notes

```
class Array
{
friend ostream &operator<<( ostream &, const Array & );
friend istream &operator>>( istream &, Array & );
public:
Array( int = 10 ); // default constructor
Array( const Array & ); // copy constructor
~Array(); // destructor
int getSize() const; // return size

const Array &operator=( const Array & ); // assignment operator
bool operator==( const Array & ) const; // equality operator

// inequality operator; returns opposite of == operator
bool operator!=( const Array &right ) const
{
return ! ( *this == right ); // invokes Array::operator==
} // end function operator!=

// subscript operator for non-const objects returns modifiable lvalue
int &operator[]( int );

// subscript operator for const objects returns rvalue
int operator[]( int ) const;
private:
int size; // pointer-based array size
int *ptr; // pointer to first element of pointer-based array
}; // end class Array

#endif
```

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://net.pku.edu.cn/~course/cs101/2007/resource/CppHowToProgram/5e/html/ch11lev1sec8.html

Self Assessment

True or False:

14. To access an array element we need the array name and the index of the element.
15. UBound() returns the number of items in the array.

4.8 Summary

- An array stores a fixed-size sequential collection of elements of the same type.
- All arrays consist of contiguous memory locations.
- The lowest address corresponds to the first element and the highest address to the last element.

Notes

- To declare an array in VB.NET, you use the Dim statement.
- An array is declared by simply using parenthesis to indicate that a variable has an index.
- A collection is typically an instance of a class and has to be declared with the New keyword to create the instance.
- You can declare a dynamic array using the ReDim statement.
- The Preserve keyword helps to preserve the data in an existing array, when you resize it.
- The string split function returns array of strings which are the splits of the given string it is delimited by the given System.Char array.
- The Join method is used when you want to join the elements of an array back together again.
- Erase statement is used to release array variables and deallocate the memory used for their elements.
- System. Array Class provides methods for creating, manipulating, searching, and sorting arrays.

4.9 Keywords

Array: It stores a fixed-size sequential collection of elements of the same type.

Collection: It is typically an instance of a class and has to be declared with the New keyword to create the instance.

Dim: It is used to declare an array in VB.NET.

Erase Statement: It is used to release array variables and deallocate the memory used for their elements.

Join( ): It is used when you want to join the elements of an array back together again.

Preserve: It helps to preserve the data in an existing array, when you resize it.

ReDim: You can declare a dynamic array using this statement.

Split( ): It returns array of strings which are the splits of the given string it is delimited by the given System.Char array.

4.10 Review Questions

1. What is an array?
2. Why do we need an array?
3. Write a note on fixed size arrays.
4. Differentiate between fixed and dynamic size arrays
5. Why do we use ReDim?
6. Explain the split( ) function with an example.
7. What is the purpose of using the join( ) function?
8. What do you mean by multidimensional arrays?
9. Explain the System.Array class.
10. Why do we use the erase statement?

Notes

Answers: Self Assessment

1.	Array	2.	Contiguous
3.	Dim	4.	Collection
5.	False	6.	True
7.	False	8.	True
9.	ReDim	10.	Preserve
11.	Erase	12.	Two
13.	Length	14.	True
15.	False		

4.11 Further Readings

Books

Learning Visual Basic .NET, Jesse Liberty, O'Reilly Media, Inc.

Object-oriented Programming with Visual Basic.Net, Hamilton.

Programming Visual Basic .NET, J. Liberty.

VB .NET Language in a Nutshell, Steven Roman, Ron Petrusha, Paul Lomax.

Online links

http://patorjk.com/programming/tutorials/vbarrays.htm

http://msdn.microsoft.com/en-IN/library/5c1seyzc%28v=vs.71%29.aspx

http://www.jblearning.com/samples/0763724785/ch02_bronson.pdf

http://vb.net-informations.com/framework/framework_tutorials.htm

Notes

Unit 5: Built-in Functions

CONTENTS

Objectives

After studying this unit, you will be able to:

- Discuss the built-in functions
- Explain string class
- Understand the conversion functions
- Discuss miscellaneous functions
- Explain the types subroutines and functions

Introduction

A procedure is referred to as built-in if it shipped with its programming language. To make your job a little easier, VB.NET comes equipped with many procedures that you can use right away in your program. Based on this, before creating your own procedure, first check whether the functionality you are looking is already implementing in one of the available procedures because those that ship with VB.NET are highly reliable and should be preferred. Before using a built-in procedure, you must of course be familiar with it. This comes either by consulting the

Notes

documentation or by experience. This means that you must know its name, its argument(s), its return value, and its role. The Microsoft Visual Basic (.NET) programming language provides one of the richest set of functions of a library. In fact, it is the richest of the .NET-based languages, giving you access to functions that are not directly available to other languages such as C# or C++/CLI. Because there so many of those functions, we will review only the most usually used. Eventually, when necessary, in other units, we may introduce new ones.

5.1 Visual Basic .NET Functions

A function is similar to a normal procedure but the main purpose of the function is to accept a certain input from the user and return a value which is passed on to the main program to finish the execution. There are two types of functions, the built-in functions (or internal functions) and the functions created by the programmers. The general format of a function is:

```
FunctionName (arguments)
```

The arguments are values that are passed on to the function.

We are going to learn two very basic but useful internal functions of Visual basic.NET, i.e. the MsgBox( ) and InputBox ( ) functions.

5.1.1 MsgBox() Function

The objective of MsgBox is to produce a pop-up message box and prompt the user to click on a command button before he/she can continues. This format is as follows:

```
yourMsg=MsgBox (Prompt, Style Value, Title)
```

The first argument, Prompt, will display the message in the message box. The Style Value will determine what type of command buttons appear on the message box. The Title argument will display the title of the message board.

Table 5.1: Styles of a Msgbox( )

Style Value	Name Constant	Buttons Displayed
0	vbOkOnly	OK button
1	vbOKCancel	Ok and Cancel buttons
2	vbAbortRetryIgnore	Abort, Retry and Ignore buttons
3	vbYesNoCancel	Yes, No and Cancel buttons
4	vbYesNo	Yes and No buttons
5	vbRetryCancel	Retry and Cancel buttons

Source: http://www.vbtutor.net/lesson10.html

We can use named constant in place of integers for the second argument to make the programs more readable.

Example:

```
yourMsg=MsgBox( "Click OK to Proceed", 1, "Startup Menu")
yourMsg=Msg("Click OK to Proceed". vbOkCancel,"Startup Menu")
```

yourMsg is a variable that holds values that are returned by the MsgBox ( ) function. The values are determined by the type of buttons being clicked by the users. It has to be declared as Integer data type in the procedure or in the general declaration section.

Notes

Table 5.2: Return Values of a Msgbox( ) Function

Value	Name Constant	Button Clicked
1	vbOk	Ok button
2	vbCancel	Cancel button
3	vbAbort	Abort button
4	vbRetry	Retry button
5	vbIgnore	Ignore button
6	vbYes	Yes button
7	vbNo	No button

Source: http://www.vbtutor.net/lesson10.html

5.1.2 InputBox() Function

An InputBox( ) function will display a message box where the user can enter a value or a message in the form of text. The format is:

```
myMessage=InputBox(Prompt, Title, default_text, x-position, y-position)
```

myMessage is a variant data type but typically it is declared as string, which accept the message input by the users. The arguments are explained as follows:

- ***Prompt*** **-** The message displayed normally as a question asked.
- ***Title*** **-** The title of the Input Box.
- ***default-text*** **-** The default text that appears in the input field where users can use it as his intended input or he may change to the message he wish to key in.
- ***x-position and y-position*** **-** the position or the coordinate of the input box.

Self Assessment

Fill in the blanks:

1. The main purpose of the ..... is to accept a certain input from the user and return a value which is passed on to the main program to finish the execution.
2. The objective of ..... is to produce a pop-up message box and prompt the user to click on a command button before he/she can continues.
3. An ..... function will display a message box where the user can enter a value or a message in the form of text.

5.2 String Class

In VB.Net you can use strings as array of characters, however, more common practice is to use the String keyword to declare a string variable. The string keyword is an alias for the System.String class.

5.2.1 Creating an Object from a Class

You can create string object using one of the following methods:

- By assigning a string literal to a String variable
- By using a String class constructor

Notes

- By using the string concatenation operator (+)
- By retrieving a property or calling a method that returns a string
- By calling a formatting method to convert a value or object to its string representation

Example:

```
Sub Main()
Dim fname, lname, fullname, greetings As String
fname = "Abc"
lname = "Def"
fullname = fname + " " + lname
Console.WriteLine("Full Name: {0}", fullname)

Dim letters As Char() = {"H", "e", "l", "l", "o"} // Using string
constructor

greetings = New String(letters)
Console.WriteLine("Greetings: {0}", greetings)

Dim sarray() As String = {"Hello", "From", "VB.NET"} // Methods returning
String
Dim message As String = String.Join(" ", sarray)
Console.WriteLine("Message: {0}", message)

// Formatting method to convert a value
Dim waiting As DateTime = New DateTime(2012, 12, 12, 17, 58, 1)
Dim chat As String = String.Format("Message sent at {0:t} on {0:D}",
waiting)
Console.WriteLine("Message: {0}", chat)
Console.ReadLine()

End Sub
```

When the above code is compiled and executed, it produces following result:

```
Full Name: Abc Def
Greetings: Hello
Message: Hello From VB.NET
Message: Message sent at 5:58 PM on Sunday, May 12, 2013
```

5.3 Conversion Functions

VB.NET supports many inbuilt conversion functions that handle the conversion of date and time, character strings, numbers and more. Let us discuss them in detail.

5.3.1 Type Conversion Functions

Sometimes we need to change the type of a variable as the program proceeds. Many a times the compiler itself effects the change. In some cases, we need to explicitly effect the change. The former type of conversion is referred to as implicit conversion. The later type is referred to as explicit conversion. Implicit conversions happen when the change effected does not change the value of the variable and there is no loss of data. Explicit conversions are a must when the change is from a large type to a smaller type.

Notes

You may recall that when studying data types, we saw that each had a corresponding function used to convert a string value or an expression to that type. As a reminder, the general syntax of the conversion functions is:

ReturnType = *FunctionName*(*Expression*)

The *Expression* could be of any kind. For example, it could be a string or expression that would produce a value such as the result of a calculation. The conversion function would take such a value, string, or expression and attempt to convert it. If the conversion is successful, the function would return a new value that is of the type specified by the *ReturnType* in our syntax.

The conversion functions are as follows:

Table 5.3: Conversion Function

Function		
Name	**Return Type**	**Description**
CBool	Boolean	Converts an expression into a Boolean value
CByte	Byte	Converts an expression into Byte number
CDate	Date	Converts and expression into a date or time value
CDbl	Double	Converts an expression into a flowing-point (decimal) number
CInt	Integer	Converts an expression into an integer (natural) value
CCur	Currency	Converts an expression into a currency (monetary) value
CLng	Long	Converts an expression into a long integer (a large natural) number
CSng	Single	Converts an expression into a flowing-point (decimal) number
CStr	String	Converts an expression into a string

Source: http://www.functionx.com/vbasic/Lesson27.htm

Task Use different functions in VB.Net and show their working.

5.3.2 Val Function

It returns the numbers contained in a string as a numeric value of appropriate type.

Public Overloads Function Val(ByVal InputStr As String) As Double

-or-

Public Overloads Function Val(ByVal Expression As Object) As Double

-or-

Public Overloads Function Val(ByVal Expression As Char) As Integer

The InputStr parameter is a required one. It can be any valid String expression, Object variable, or Char value. If Expression is of type Object, its value must be convertible to String or an ArgumentException error occurs.

The Val function stops reading the string at the first character it cannot recognize as part of a number. Symbols and characters that are often considered parts of numeric values, such as dollar signs and commas, are not recognized. However, the function recognizes the radix prefixes &O (for octal) and &H (for hexadecimal). Blanks, tabs, and linefeed characters are stripped from the argument.

Notes

The following call returns the value 1615198.

```
Val(" 1615 198th Street N.E.")
```

The following call returns the decimal value -1.

```
Val("&HFFFF")
```

Note The Val function recognizes only the period (.) as a valid decimal separator. When different decimal separators are used, as in international applications, use CDbl or CInt instead to convert a string to a number.

Example: The following example uses the Val function to return the numbers contained in each string. Val stops converting at the first character that cannot be interpreted as a numeric digit, numeric modifier, numeric punctuation, or white space.

```
Dim valResult As Double
' The following line of code sets valResult to 2457.
valResult = Val("2457")
' The following line of code sets valResult to 2457.
valResult = Val(" 2 45 7")
' The following line of code sets valResult to 24.
valResult = Val("24 and 57")
```

The Str(or Str$) Function

It is used to return string equivalent for the specified integer.

Public Shared Function Str(ByVal Number As Object) As String

In the above syntax Number specifies a valid numeric expression.

Example:

```
Sub Main()
  Console.WriteLine('String returned for 5.5 is::' & Str(5.5))
  Console.WriteLine('String returned for -5.5 is::' & Str(-5.5))
  Console.ReadLine()
End Sub
```

```
Output:

String returned for 5.5 is:: 5.5
String returned for -5.5 is::-5.5
```

In the above example, preceding space is left for positive characters, but a minus is displayed for negative integers. Thus Str string function can be used.

Conversion Classes

The Convert class converts a base data type to another base data type. Its syntax is:

```
Public NotInheritable Class Convert
```

Notes

The static methods of the Convert class are used to support conversion to and from the base data types in the .NET Framework. The supported base types are Boolean, Char, SByte, Byte, Int16, Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Decimal, DateTime and String.

Example: The following example demonstrates some of the conversion methods in the Convert class, including ToInt32, ToBoolean, and ToString.

```
Dim dNumber As Double
dNumber = 23.15

Try
 // Returns 23
 Dim iNumber As Integer
 iNumber = System.Convert.ToInt32(dNumber)
Catch exp As System.OverflowException
 System.Console.WriteLine("Overflow in double to int conversion.")
End Try

 // Returns True
Dim bNumber As Boolean
bNumber = System.Convert.ToBoolean(dNumber)

 // Returns "23.15"
Dim strNumber As String
strNumber = System.Convert.ToString(dNumber)

Try
 // Returns '2'
 Dim chrNumber As Char
 chrNumber = System.Convert.ToChar(strNumber.Chars(1))
Catch exp As System.ArgumentNullException
 System.Console.WriteLine("String is null.")
Catch exp As System.FormatException
 System.Console.WriteLine("String length is greater than 1.")
End Try

// System.Console.ReadLine() returns a string and it must be converted.
Dim newInteger As Integer
newInteger = 0
Try
 System.Console.WriteLine("Enter an integer:")
 newInteger = System.Convert.ToInt32(System.Console.ReadLine())
Catch exp As System.ArgumentNullException
 System.Console.WriteLine("String is null.")
Catch exp As System.FormatException
 System.Console.WriteLine("String does not consist of an " + _
 "optional sign followed by a series of digits.")
Catch exp As System.OverflowException
 System.Console.WriteLine("Overflow in string to int conversion.")
End Try

System.Console.WriteLine("Your integer as a double is {0}", _
 System.Convert.ToDouble(newInteger))
```

Notes

Self Assessment

Fill in the blanks:

4. In VB.NET you can use ..... as array of characters.
5. ..... Functions are used to change the type of a variable as the program proceeds.
6. ..... conversions happen when the change effected does not change the value of the variable and there is no loss of data.
7. ..... conversions are a must when the change is from a large type to a smaller type.
8. ..... Function returns the numbers contained in a string as a numeric value of appropriate type.
9. The ..... Function is used to return string equivalent for the specified integer.
10. The ..... class converts a base data type to another base data type.

5.4 Miscellaneous Functions

VB.NET also supports some miscellaneous functions like CallByName Function, Choose Function, Command Function, CreateObject Function, etc.

5.5 Subroutines and Functions

We will now discuss the subroutines and functions one by one in detail.

5.5.1 Subroutine

A subroutine is a block of statements that carries out a well-defined task. The block of statements is placed within a set of Sub. . .End Sub statements and can be invoked by name.

Example:

The following subroutine displays the current date in a message box and can be called by its name, ShowDate():

```
Sub ShowDate()
MsgBox(Now().ToShortDateString)
End Sub
```

Normally, the task performed by a subroutine is more complicated than this; but even this simple subroutine is a block of code isolated from the rest of the application. The statements in a subroutine are executed, and when the End Sub statement is reached, control returns to the calling program. It's possible to exit a subroutine prematurely by using the Exit Sub statement. All variables declared within a subroutine are local to that subroutine. When the subroutine exits, all variables declared in it cease to exist. Most procedures also accept and act upon arguments. The ShowDate() subroutine displays the current date in a message box. If you want to display any other date, you have to implement it differently and add an argument to the subroutine:

```
Sub ShowDate(ByVal birthDate As Date)
MsgBox(birthDate.ToShortDateString)
End Sub
```

Notes

birthDate is a variable that holds the date to be displayed; its type is Date. The ByVal keyword means that the subroutine sees a copy of the variable, not the variable itself. What this means practically is that the subroutine can't change the value of the variable passed by the calling application. To display the current date in a message box, you must call the ShowDate() subroutine as follows from within your program:

```
ShowDate()
```

To display any other date with the second implementation of the subroutine, use a statement like the following:

```
Dim myBirthDate = #2/9/1960#
ShowDate(myBirthDate)
```

Or, you can pass the value to be displayed directly without the use of an intermediate variable:

```
ShowDate(#2/9/1960#)
```

If you later decide to change the format of the date, there's only one place in your code you must edit: the statement that displays the date from within the ShowDate() subroutine.

5.5.2 Functions

A function is similar to a subroutine, but a function returns a result. Because they return values, functions — like variables — have types. The value you pass back to the calling program from a function is called the return value, and its type must match the type of the function. Functions accept arguments, just like subroutines. The statements that make up a function are placed in a set of Function…End Function statements, as shown here:

```
Function NextDay() As Date
Dim theNextDay As Date
theNextDay = Now.AddDays(1)
Return theNextDay
End Function
```

The Function keyword is followed by the function name and the As keyword that specifies its type, similar to a variable declaration. AddDays is a method of the Date type, and it adds a number of days to a Date value. The NextDay() function returns tomorrow's date by adding one day to the current date. NextDay() is a custom function, which calls the built-in AddDays method to complete its calculations. The result of a function is returned to the calling program with the Return statement, which is followed by the value you want to return from your function. This value, which is usually a variable, must be of the same type as the function. In our example, the Return statement happens to be the last statement in the function, but it could appear anywhere; it could even appear several times in the function's code. The first time a Return statement is executed, the function terminates, and control is returned to the calling program. You can also return a value to the calling routine by assigning the result to the name of the function. The following is an alternate method of coding the NextDay() function:

```
Function NextDay() As Date
NextDay = Now.AddDays(1)
End Function
```

Notice that the result of the calculation has been assigned to the function's name directly and didn't use a variable. This assignment, however, doesn't terminate the function like the Return statement. It sets up the function's return value, but the function will terminate when the End Function statement is reached, or when an Exit Function statement is encountered. Similar to variables, a custom function has a name that must be unique in its scope (which is also true for subroutines, of course). If you declare a function in a form, the function name must be unique in

Notes

the form. If you declare a function as Public or Friend, its name must be unique in the project. Functions have the same scope rules as variables and can be prefixed by many of the same keywords. In effect, you can modify the default scope of a function with the keywords Public, Private, Protected, Friend, and Protected Friend. In addition, functions have types, just like variables, and they're declared with the As keyword.

Suppose that the function CountWords() counts the number of words, and the function CountChars() counts the number of characters in a string. The average length of a word could be calculated as follows:

```
Dim longString As String, avgLen As Double
longString = TextBox1.Text
avgLen = CountChars(longString) / CountWords(longString)
```

The first executable statement gets the text of a TextBox control and assigns it to a variable, which is then used as an argument to the two functions.When the third statement executes, Visual Basic first calls the functions CountChars() and CountWords() with the specified arguments, and then divides the results they return.

You can call functions in the same way that you call subroutines, but the result won't be stored anywhere. For example, the function Convert() might convert the text in a text box to uppercase and return the number of characters it converts. Normally, you'd call this function as follows:

```
nChars = Convert()
```

If you don't care about the return value — you only want to update the text on a TextBox control — you would call the Convert() function with the following statement:

```
Convert()
```

Case Study

Implementing Subroutines with Arrays

Introduction

We're going to implement a function that passes an array. This function is called findMin which finds the minimum value in an array.

findMin

Here's a simple C function that finds the minimum value of an array of **n** values passed in, and returns it.

```
   int min = arr[ 0 ] ;
   int i ;
   for ( i = 0 ; i < n ; i++ )
     {
         if ( arr[ i ] < min )
            min = arr[ i ] ;
     }
   return min ;
}
```

Arrays are really just pointers, and in MIPS, pointers are 32 bits (at least, MIPS R2000/R3000—later MIPS CPUs use 64 bits of addresses). Thus, we can pass the pointers using the argument registers since those registers can store 32 bits.

Contd...

Notes

It's often quite convenient that registers store the same number of bits as addresses.

Implementation

Here's the implementation in MIPS. We assume arr is stored in $a0 and n is stored in $a1. We use $t0 to store min and $t1 to store i. Notice this is a leaf procedure. So, we won't use the stack. Also notice that loading a word at the address stored in $a0 access arr[0].

```
findMin:    lw $t0, 0($a0)          # min = arr[0]
            li $t1, 0               # i = 0
LOOP:       bge $t1, $a1, END       # branch ! (i < n)
            add $t2, $t1, $t1       # t2 = 2 * i
            add $t2, $t2, $t2       # t2 = 4 * i
            add $t2, $t2, $a0       # t2 = arr + (4 * i)
            lw  $t3, 0( $t2 )       # t3 = arr[ i ]
            bge $t3, $t0, INCR      # branch ! (arr[i] < min)
            move $t0, $t3           # min = arr[i]
INCR:       addi $t1, $t1, 1        # i++
            j LOOP                  # back to top of loop
END:        move $v0, $t0           # retval = min
            jr $ra                  # return
```

The awkward part is computing the address of the array element. Recall that & arr[i] is arr + (sizeof(int) * i) where we assume sizeof(int) is 4 bytes. Thus, we must compute 4 * i. There are two ways to do this. Multiply by 4 (which involves some complications of its own) or add the number to itself twice (which seems awkward).

Also notice where the branch statement for the if-statement goes. If the condition is false, it jumps to the increment.

Making the Call

How do we call findMin? If arr is a label which is conveniently declared in the data segment, then all we do is something like:

```
la $a0, arr # set arg 0
li $a1, 10 # set arg 1
jal findMin
move $t0, $v0 # save return value to $t0
```

findMin, version 2

This is the "same" function as before, at least, in the purpose of the function. It finds the minimum value in an array and returns it.

```
  int min = * arr ;
  int i ;
  for ( i = 0 ; i < n ; i++ )
    {
        if ( *arr < min )
           min = * arr ;
         arr++ ;   // POINTER ARITHMETIC: Move arr forward 1 element
    }
  return min ;
}
```

Contd...

Notes

The difference in this version is that we treat the **arr** as a pointer variable, and move the pointer forward one element at a time.

In the first version, findMin, we did not alter arr's value. Instead, each iteration, we computed the offset from the start of the array to element i of the array. We summed the offset and arr and storing the sum into a temporary register. Then, we loaded the array element from the address stored in this temporary register.

As you can see, the first version is more complicated to explain and implement in MIPS, even though it's somewhat easier to understand the array version than the pointer version (because most people understand arrays better than pointers, even though, in C, the two are related to one another).

Implementation

Here's the implementation in MIPS. We assume arr is stored in $a0 and n is stored in $a1. We use $t0 to store min. This version is somewhat simpler because we modify arr, which we did not do before, and we do not need to do the more difficult computations of addresses as in the previous version.

```
findMinTwo: lw $t0, 0($a0)          # min = arr[0]
            li $t1, 0               # i = 0
LOOP:       bge $t1, $a1, END       # branch ! (i < n)
            lw  $t2, 0( $a0 )       # t2 = * arr
            bge $t2, $t0, INCR      # branch ! (*arr < min)
            move $t0, $t2           # min = * arr
INCR:       addi $a0, $a0, 4        # Move arr forward 1 element
            addi $t1, $t1, 1        # i++
            j LOOP                  # back to top of loop
END:        move $v0, $t0           # retval = min
            jr $ra                  # return
```

This code is shorter and more efficient than the array version. This is one reason why a good knowledge of pointers can help you write more efficient code. However, realize the tradeoffs between efficent code, and code that's easy to follow and maintain.

Summary

Implementing a function with an array is the first time we've actually used an address from memory and used the **lw** instruction. While compilers allow us to access the **i**th element of an array conveniently (via pointer arithmetic), we don't get such conveniences in assembly language. We have to compute the offset explicitly as in findMin.

If you don't mind moving the pointer (which you can do, if it's passed as a parameter), then you can avoid computing the offset, as in findMinTwo.

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Mips/case_arr.html

Self Assessment

Notes

True or False:

11. A subroutine is a block of statements that carries out a well-defined task.
12. A function is placed within a set of Sub…End Sub statements.
13. Subroutine cannot be invoked by name.
14. A function does not returns a result.
15. You can call functions in the same way that you call subroutines, but the result won't be stored anywhere.

5.6 Summary

- A procedure is referred to as "built-in" if it shipped with its programming language.
- A function is similar to a normal procedure but the main purpose of the function is to accept a certain input from the user and return a value which is passed on to the main program to finish the execution.
- The objective of MsgBox is to produce a pop-up message box and prompt the user to click on a command button before he/she can continues.
- An InputBox( ) function will display a message box where the user can enter a value or a message in the form of text.
- In VB.Net you can use strings as array of characters.
- Type Conversion Functions are used to change the type of a variable as the program proceeds.
- Implicit conversions happen when the change effected does not change the value of the variable and there is no loss of data.
- Explicit conversions are a must when the change is from a large type to a smaller type.
- Val Function returns the numbers contained in a string as a numeric value of appropriate type.
- The Str(or Str$) Function is used to return string equivalent for the specified integer.
- The Convert class converts a base data type to another base data type.
- A function is similar to a subroutine, but a function returns a result.

5.7 Keywords

Convert class: It converts a base data type to another base data type.

Explicit conversions: They are a must when the change is from a large type to a smaller type.

Implicit conversions: They happen when the change effected does not change the value of the variable and there is no loss of data.

InputBox( ) : It will display a message box where the user can enter a value or a message in the form of text.

MsgBox: It is used to produce a pop-up message box and prompt the user to click on a command button before he/she can continues.

Notes

Procedure: It is referred to as "built-in" if it shipped with its programming language.

Str(or Str$) Function: It is used to return string equivalent for the specified integer.

Strings: It is an array of characters.

Type Conversion Functions: They are used to change the type of a variable as the program proceeds.

Val Function: It returns the numbers contained in a string as a numeric value of appropriate type.

5.8 Review Questions

1. What is a procedure?
2. Explain the MsgBox() function with an example.
3. Differentiate between a msgbox and an inputbox.
4. Why do we need a String class?
5. How do we create an object from a String class?
6. List some types of conversion functions with examples.
7. Write a note on Val function.
8. What is Convert class?
9. What is a subroutine?
10. Differentiate between a subroutine and a function.

Answers: Self Assessment

1.	Function	2.	Msgbox
3.	InputBox	4.	Strings
5.	Type Conversion	6.	Implicit
7.	Explicit	8.	Val
9.	Str	10.	Convert
11.	True	12.	False
13.	False	14.	False
15.	True		

5.9 Further Readings

Books

Object-oriented Programming with Visual Basic.Net, Hamilton.

Programming Visual Basic .NET, J. Liberty.

Visual Basic .NET, Shirish Chavan, Pearson Education India.

Visual Basic.NET Black Book, Steven Holzner.

Notes

Online links

http://uet.vnu.edu.vn/tltk/Learning/File_PDF/CacHamTrongVB.NET.pdf

http://www.yevol.com/en/visualbasic/Lesson08.htm

http://visualbasic.w3computing.com/vb2008/3/vb-built-in-functions.php

http://msdn.microsoft.com/en-us/library/k7beh1x9%28v=vs.80%29.aspx

Notes

Unit 6: Classes and Object in VB.NET

CONTENTS

Objectives

After studying this unit, you will be able to:

- Understand classes, objects and methods
- Explain constructors
- Discuss the properties and indexers
- Elaborate the concept of inheritance

Introduction

The original versions of Microsoft Visual Basic provided a mechanism for defining data structures in a user-defined type (UDT). A UDT encapsulates the data, but not the processing associated with that data. Processing was defined in global standard modules, often called BAS modules because of their .bas extension. The release of Visual Basic 4 dawned a new age for Visual Basic developers. Visual Basic took its first steps toward becoming an object-oriented programming (OOP) language by providing object-oriented features such as class modules. A class module defines data as properties and the processing associated with that data as methods. By defining a class for each business entity, encapsulating data in properties and processing in methods, Visual Basic developers had object-based development. As Visual Basic evolved from version 4 to version 6, Visual Basic developers expanded their knowledge of OO to include component-based development (CBD) techniques. With CBD, Visual Basic developers

Notes

could build complete three-tiered applications for Microsoft Windows® and the Web. This type of development was so common that Microsoft provided a design pattern known as the Microsoft DNA architecture. Visual Basic .NET provides another leap in Visual Basic development capabilities and features and provides for true object-oriented programming, as detailed in this article.

For a programming language to be a true OOP language, the language must meet the following criteria:

- ***Abstraction:*** Abstraction manages the complexities of a business problem by allowing you to identify a set of objects involved with that business problem.
- ***Encapsulation:*** Encapsulation hides the internal implementation of an abstraction within the particular object.
- ***Polymorphism:*** Polymorphism provides for multiple implementations of the same method. For example, different objects can have a Save method, each of which perform different processing.
- ***Inheritance:*** The excitement of Visual Basic .NET lies in inheritance. Visual Basic 5 introduced the concept of interface inheritance, which allows you to reuse the interface of a class, but not its implementation. Visual Basic .NET provides for true implementation inheritance whereby you can reuse the implementation of a class.

6.1 Using Classes, Object and Methods

Visual Basic .NET is not Visual Basic 6 with inheritance tacked onto it. Rather, Visual Basic .NET has been entirely rewritten to be fully object-oriented. In fact, everything in Visual Basic .NET can be treated as an object. Yes, even your strings and integers can be accessed as objects in Visual Basic .NET.

To demonstrate this, start a new Visual Basic .NET console application project. In the Main subroutine, enter this code:

```
Dim i As Integer
MsgBox(i.MinValue)
```

The first hint that your integer is treated as an object is the list of properties and methods that appear when you type the dot after the **i**. Select one of the properties, such as the MinValue shown in this example. Then run the application and you will get a message box containing the value of the selected integer property.

6.1.1 Class

The basic purpose of a class has not changed in Visual Basic .NET. You still create classes for your business objects and for any supporting objects that you may need for your application. The primary changes from Visual Basic 6 to Visual Basic .NET involve syntax and some new features.

In Visual Basic 6, you create a class by creating a class module: one class, one class module. This is no longer the case in Visual Basic .NET. You can create any number of classes within a single code file. You can even create classes within classes. But let's start out with a simple example.

Adding a class to a Visual Basic .NET project is very similar to Visual Basic 6. However, instead of getting an empty code file, your class will appear with the following code:

```
Public Class CCustomer
End Class
```

Notes

Note The current Microsoft convention is to define the class names without a prefix. Following that convention, this class name would be Customer instead of CCustomer. While this convention is more user-friendly when creating objects using retail products such as Microsoft Word and Excel, maintenance and support of enterprise systems can benefit from the additional information that a prefix provides.

If you want to add a second class to the same file, just add another class statement:

```
Public Class CCustomer
End Class
Public Class CContact
End Class
```

Note Normally, a class should be defined in its own code file. Only put classes together if they are tightly coupled. For example, invoice and invoice line item could be two classes within one code file because you would normally never use invoice line item without invoice. If you would use customer contacts (CContact) separate from customers then the CContact class should be separate from the CCustomer class.

Create a Class

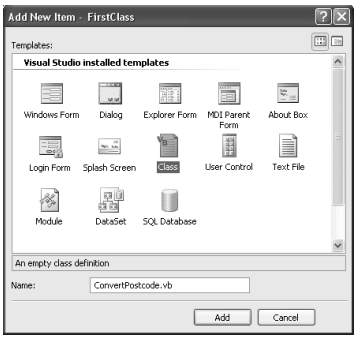
To create a class in VB.NET, follow the steps below:

- Start a new VB.NET project
- Add a Textbox to your form, and leave it on the default Name, TextBox1
- Change the Text Property to ts1 4jh (make sure the letters are lowercase and not upper, because our object will convert it.)
- Add a Button to your form

Once you have a new form with a Textbox and a Button on it, you need to add a Class. This is quite easy. It's just like adding a Module. In fact, they look exactly the same.

- So from the VB menu bar, click on Project
- From the drop down menu, click Add Class
- You'll get this dialogue box popping up in version 2008:

Figure 6.1: Add New Item



Source: http://www.homeandlearn.co.uk/net/nets11p2.html

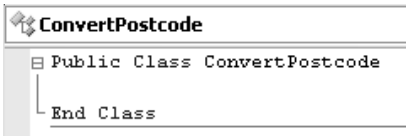
Notes

The Class Template on the right will already be selected. The thing you need to change is the Name at the bottom. The default Name is Class1.vb. This is not terribly descriptive, and you'll have great problems working out what this class does, a few months down the line.

Change the Name from Class1.vb to ConvertPostcode.vb. Then click the Open button.

When the code window for the class opens up, it will look like this:

Figure 6.2: Code Window



Source: http://www.homeandlearn.co.uk/net/nets11p2.html

As you can see, there's not a great deal to look at! All we have is the Public Class … End Class code stub. The name of our Class is also there. But the code is in a separate window, and has a tab all to itself. It's this tab full of code that you reuse and turn into an object.

What we have to do now is add the code that does the work - converts our postcode. But we can't just write this:

```
Dim ConvertPostcode As String
ConvertPostcode = StrConv( TextBox1.Text, VbStrConv.UpperCase )
TextBox1.Text = ConvertPostcode
```

That would be all right for the button on our Form. But it's not all right for our Class. When you're designing a Class, your code has to go inside of things like Functions and Subs. When you set up a Function or Sub, you're actually creating Methods for your objects A Method is code that actually does something, that performs an action. Converts a postcode in our case.

6.1.2 Creating an Object from a Class

In order to use a class, you first need to create an object from the class. In Visual Basic 6, the recommended syntax for creating an object from a class is:

```
Private m_oCustomer as CCustomer
Set m_oCustomer = New CCustomer
```

This syntax is almost identical in Visual Basic .NET. Since everything is basically an object in Visual Basic .NET, there is no need for two different types of assignment, so there is no longer a need for the Set keyword:

```
Private m_oCustomer as CCustomer
m_oCustomer = New CCustomer()
```

Notice the parenthesis when creating the object. If you defined a parameterized constructor for the class, you can pass the parameter(s) to the constructor within the parentheses:

```
m_oCustomer = New CCustomer("Acme Corporation")
```

With Visual Basic .NET, you can combine the object variable declaration and the object creation using the New keyword:

```
Private m_oCustomer As CCustomer = New CCustomer()
```

The object is created when this declaration is executed. The shorthand form of this syntax is:

```
Private m_oCustomer As New CCustomer()
```

Notes

If you need to pass parameters, this syntax becomes:

```
Private m_oCustomer As New CCustomer("Acme Corporation")
```

You cannot put a Try... Catch block for error handling around a module-level declaration. This restriction makes this style of object creation less useful. In production-quality applications, you may want to stick to the tried and true approach of first declaring the module-level variable and then creating an instance with the New keyword in order to support full-featured error handling.

In the declarations section:

```
Private m_oCustomer as CCustomer
```

Within a routine:

```
Try
      m_oCustomer = New CCustomer()
Catch e As Exception
      Debug.WriteLine(e.Message)
End Try
```

Continuing with the example, select your favorite style of object creation and add it to the Sub Main.

When you have finished using an object, you can call the object's Dispose method to free the resources used by the object, assuming that a Dispose method was implemented for the object. You can then set the object variable to Nothing:

```
m_oCustomer.Dispose()
m_oCustomer = Nothing
```

The object is then destroyed when the garbage collector detects that the object is no longer used.

Tip Unlike Visual Basic 6, an object in .NET is not destroyed the moment that the object variable is set to Nothing. Rather, it will be destroyed when the garbage collector detects and destroys it. The garbage collector will destroy an orphaned object, whether or not you set the object variable to Nothing.

At this point, you should be able to execute the sample application and see the debug messages appear in the Output window.

6.1.3 Create a Method

You can also then add properties or methods to the class. As with Visual Basic 6, you normally define a property by declaring a private variable and public Property procedures. In Visual Basic .NET, you would define a Name property as follows:

```
Private m_sName As String
Property Name() As String
    Get
         Return m_sName
    End Get
     Set(ByVal Value As String)
         m_sName = Value
    End Set
End Property
```

There are only two types of Property procedures in Visual Basic .NET, Get and Set. The Get procedure retrieves the property value from the class and the Set procedure assigns the property. Visual Basic 6 provided a property Let statement that handled intrinsic data types while the Set

Notes

statement worked with objects. Now that everything in Visual Basic .NET is basically an object, there is no need for the Let statement. Notice that the syntax for a property procedure is also changed. Both the Get and Set are contained within one property statement. No more possibility of a mismatch in data types between property Get and Set. This makes these statements easier to maintain. In a three-tiered or N-tiered application, your classes may be stateless, meaning that they have no properties. This provides more efficient use of your classes within middle-tier components.

The syntax for a simple method is nearly identical to prior versions of Visual Basic. The only difference you may notice is the Return keyword. You can use Return to return a value from a function instead of using the function name. The following example demonstrates a simple method:

```
Public Function SayHello() As String
    If Name <> "" Then
        Return "Hello " & Name
    Else
        Return "Hello World"
    End If
End Function
```

Self Assessment

Fill in the blanks:

1. ..... manages the complexities of a business problem by allowing you to identify a set of objects involved with that business problem.
2. ..... hides the internal implementation of an abstraction within the particular object.
3. ..... provides for multiple implementations of the same method.
4. ..... allows you to reuse the interface of a class, but not its implementation.
5. In order to use a class, you first need to create an ..... from the class..

6.2 Constructor

In Visual Basic 6, when you create an instance of a class the Initialize event is generated. You can put code into the Initialize event to initialize the object. For example, you may want to define default object data, open database connections, or create related objects. However, you cannot pass anything to the Initialize event. This makes it difficult to initialize the object with specific parameters.

Visual Basic .NET introduces true constructors that are executed whenever a new instance of the class is created. These constructors are defined with a subroutine named New.

```
Public Sub New()
    ' Perform initialization
    Debug.WriteLine("I am alive")
End Sub
```

You can pass data to a constructor for more flexibility and power in initializing the object. Constructors with parameters are called *parameterized constructors*. For example:

```
Public Sub New(ByVal sName As String)
    ' Assign the name
```

Notes

```
        Name = sName
         'Other initialization
         Debug.WriteLine(Name & " is alive")
    End Sub
```

In this example, the customer name is passed in to the constructor. That name is then used to initialize the Name property defined with the Property procedure.

Both of these constructors can be define for one class. Actually, any number of constructors can be defined for a class as long as they each have different parameters. This feature is called *overloading*. The appropriate constructor is called based on the data passed to the constructor. You do not have to define a constructor. If you don't create one, a default constructor is used.

Shared Constructors

Shared constructors are used to initialize the **shared variables** of a type. Shared variables are created using the Shared keyword and store values that can be shared by all the instances of a class. Shared constructors have an **implicit** public access. A shared constructor will not run more than once during a single execution of a program.

The following example is an illustration of the shared constructor.

```
Public Class class1
   Shared x As Integer

   Shared Sub New()
          x=0
   End Sub
End Class
```

We can increment the value of a shared variable in an instance constructor to keep track of the number of instances created in a class. The following code illustrates the use of a shared variable within an instance constructor.

```
Sub New
x=x+1
MessageBox.Show("Number of instances are:" &i)
End Sub
```

To test how shared constructor works, create a form and name it as Form1 and place a button Button1.

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

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

```
Dim c1 As class1 = New class1
Dim c2 As class1 = New class1
End Sub
End Class
```

The above code illustrates the use of a shared variable within an instance constructor to keep track of the number of instances of a class.

Instance Constructor

Notes

Instance constructors are used to initialize variables that are declared with Dim, Public, Private, Friend, Protected, and Protected Friend keywords. Write the following code in the class module.

```
Public Class ItemClass
 Private ItemCode As String
 Private ItemName As String
 Private ItemDescription As String
 Private ItemCategory As String
 Private ItemPrice As Single
 Private ItemUnit As String

 Public Sub New(ByVal Category As string)
   ItemCategory = Category
 End Sub
End Class
```

In the Instance Constructor, the statement, ItemCategory = Category assigns Item Category to class variable ItemCategory.

To test how Instance constructor works, create a form and name it as Form1 and place a button Button1.

```
Public Class Form1
Inherits System.windows.Forms.Form
Private Sub Button1_Click(ByVal sender As System.Object, Byval e As
System.EvenArgs) Handles  Button1.Click
Dim objItem As New ItemClass("I")
End Sub
End Class
```

This is how we can instantiate the Item class which in turn calls the instance constructor.

Self Assessment

True or False:

6. Constructors that are executed whenever a new instance of the class is created.
7. True constructors are defined with a subroutine named New.
8. If you don't create one, no constructor is used.
9. Instance constructors are used to initialize the shared variables of a type.

6.3 Creating Properties and Indexers

In VB.NET, properties are nothing but natural extension of data fields. They are usually known as 'smart fields' in VB.NET community. We know that data encapsulation and hiding are the two fundamental characteristics of any object oriented programming language. In VB.NET, data encapsulation is possible through either classes or structures. By using various access modifiers like private, public, protected, internal etc. it is possible to control the accessibility of the class members. Usually inside a class, we declare a data field as private and will provide a set of public SET and GET methods to access the data fields. This is a good programming practice, since the data fields are not directly accessible outside the class. We must use the set/get methods to access the data fields.

Notes

Indexers, another nifty feature of VB.NET, are similar to the overloaded [] (array subscript) operator in C++. An indexer allows you to access a class instance in terms of a member array. An indexer declaration may include a set of attributes; a new modifier; a valid combination of the public, private, protected, and internal access modifiers; and one of the virtual, override, or abstract modifiers.

6.3.1 Creating Properties

You can add your own Properties to your Class. A Property, remember, is something that changes or sets a value. Examples are, setting the Text in a textbox, changing the background colour of a Form, and setting a Button to be Enabled. You can Get values from a Property or Set them. So for a Textbox, you can Set the text to appear in the textbox, or you can Get what text is inside of the textbox. You use these same words, Get and Set, when you're creating your own Properties.

Example:

Follow the steps given below to create a property:

- Add a Picture Box control to your Form
- Set the SizeMode Property of the Picture box to StretchImage
- Click on the Image Property, and add the planet.jpg image that you downloaded above
- Add two textboxes to the form. Change the Name of the first one to txtHeight, and the second one to txtWidth. Enter 300 as a the text for both textboxes
- Add two labels to the form. Set the Text of the first one to Height, and the second one to Width. Move them next to the textboxes
- Add a new button to your form. Set the Text property to "Change Height and Width"

What we'll do is to give our object the capability of setting a Height and Width property. When the object has done its work, the height and width of the picture box will change to the values from the textboxes. Off we go then.

VB needs to know that you want to set up a Property for your Class. The way you do this is type "Public Property … End Property".

Access the code for your Class. Type a few lines of space between the End Sub of your DoMessageBox Method, and the line that reads "End Class". On a new line, type the following:

```
Public Property ChangeHeight() As Integer
```

ChangeHeight is the name of our property, and it's something we made up ourselves. After a pair of round brackets, you add the type of value that will be returned (Just like a function). Here, we want to return an Integer value.

When you press the return key after typing that line, VB finishes off the rest of the code stub for you:

```
Public Property ChangeHeight() As Integer
Get
End Get
Set(ByVal Value As Integer)
End Set
End Property
```

Notes

Before the code is explained, add a new variable right at the top of your code, just below "Public Class changeHeightWidth". Add this:

```
Private intHeight As Integer
```

The Private word means that only code inside of the Class can see this variable. You can't access this code directly from a button on a Form, for example. The reason the variable is right at the top is so that other chunks of code can see and use it. But your coding window should now look something like this next image:

Figure 6.3: Code Window

```
Public Class ConvertPostcode

    Private intHeight As Integer

    Public Function DoConvert(ByVal postcode As String) As String

      Dim ConvertPostcode As String
      ConvertPostcode = StrConv(postcode, VbStrConv.Uppercase)
      DoConvert = ConvertPostcode

    End Function

    Public Sub DoMessageBox()

        MsgBox("Conversion Complete")

    End Sub

    Public Property ChangeHeight() As Integer

        Get

        End Get

        Set(ByVal Value As Integer)

        End Set

    End Property

End Class
```

Source: http://www.homeandlearn.co.uk/net/nets11p2.html

With the Get and Set parts, the Property stub is this:

```
Public Property PropertyName() As VariableType
End Property
```

The reason the Get and Set are there is so that you can Set a value for your property, and get a value back out.

To Set a value, the code inside of Property is this:

```
Set( ByVal Value As Integer )
End Set
```

The Set word is followed by a pair of round brackets. Inside of the round brackets is ByVal Value As Integer. The is just like a Sub, when you hand over a value to it. The name of the variable, Value, is a default name. You can change this to anything you like. The type of variable, As Integer, is also a default. You don't have to pass numbers to you property. If you want your Property to handle text you might have something like this:

```
Set( ByVal MyText As String )
```

But you couldn't do this:

```
Set( ByVal Value As Integer, ByVal MyString As String )
```

In other words, you can't pass two values to your property. You can only pass one value.

Notes

But we want to pass a number to our property. For us, this value will come from the textbox on the form. Whatever number is inside of the textbox will get handed over to our Property.

```
Set( ByVal Value As Integer )
```

But we need to use this value being handed over. We can assign it to that variable we set up at the top of the Class. So add this to your code (The new line is in bold):

```
Set(ByVal Value As Integer)
intHeight = Value
End Set
```

Whenever our Property is called into action, we're setting a Value, and then handing that value to a variable called intHeight. This is known as Writing to a Property.

To read from a Property, you use Get. This will Get a value back out of your Property. The code stub is this:

```
Get
End Get
```

You don't need any round brackets for the Get part. You're just fetching something to be read.

Add the line in bold text to your Get statement.

```
Get
ChangeHeight = intHeight
End Get
```

All you're doing here is returning a value, just like you do with a function. You're handing a value to whatever name you called your property. We called ours ChangeHeight. It's an Integer. So we can pass whatever value was stored inside of intHeight over to the variable called ChangeHeight:

```
ChangeHeight = intHeight
```

You can also use the Return keyword. Like this:

```
Get
Return intHeight
End Get
```

6.3.2 Creating Indexer

An indexer declaration specifies the element type of the indexer introduced by the declaration. Unless the indexer is an explicit interface member implementation, the type is followed by the keyword this. For an explicit interface member implementation, the type is followed by an interface type, a period (.), and the keyword this. Unlike other members, indexers do not have user defined names. The formal parameter list of an indexer corresponds to that of a method, with two differences: at least one parameter must be specified, and the ref and out parameter modifiers are not permitted. The accessors specify the executable statements associated with reading and writing indexer elements. Even though the syntax for accessing an indexer element is the same as that for an array element, an indexer element is not classified as a variable. Thus, it is not possible to pass an indexer element as a ref or out parameter. It is an error for an indexer accessor to declare a local variable with the same name as an indexer parameter. With these differences in mind, all rules defined in apply to indexer accessors as well as property accessors.

Indexers and properties, although very similar in concept, differ in the following ways:

- A property is identified by its name whereas an indexer is identified by its signature.
- A property is accessed through a simple-name or a member-access whereas an indexer element is accessed through an element-access.

Notes

- A property can be a static member whereas an indexer is always an instance member.
- A get accessor of a property corresponds to a method with no parameters whereas a get accessor of an indexer corresponds to a method with the same formal parameter list as the indexer.
- A set accessor of a property corresponds to a method with a single parameter named value whereas a set accessor of an indexer corresponds to a method with the same formal parameter list as the indexer, plus an additional parameter named value.

Example:

```
Option Strict On
 Imports System

 Public Class MyItemList
      Private strings(255) As String
     Private ctr As Integer = 0

      Public Sub New(ByVal ParamArray initialStrings( ) As String)
         Dim s As String

          For Each s In initialStrings
              strings(ctr) = s
             ctr += 1
         Next
     End Sub

      Public Sub Add(ByVal theString As String)
          If ctr >= Strings.Length Then
         Else
               Strings(ctr) = theString
             ctr += 1
         End If
     End Sub

      Default Public Property Item(ByVal index As Integer) As String
         Get
               If index < 0 Or index >= strings.Length Then
             Else
                   Return strings(index)
             End If
         End Get
          Set(ByVal Value As String)
              If index >= ctr Then
             Else
                   strings(index) = Value
             End If
         End Set
     End Property

      Public Function Count( ) As Integer
         Return ctr
```

Notes

```
     End Function
End Class

Public Class Tester
     Public Shared Sub Main( )
          Dim lbt As New MyItemList("Hello", "World")
         Dim i As Integer

           Console.WriteLine("After creation...")
         For i = 0 To lbt.Count - 1
                Console.WriteLine("lbt({0}): {1}", i, lbt(i))
        Next

         lbt.Add("W")
         lbt.Add("I")
         lbt.Add("J")
         lbt.Add("t")

           Console.WriteLine("After adding strings...")
         For i = 0 To lbt.Count - 1
                Console.WriteLine("lbt({0}): {1}", i, lbt(i))
        Next

         Dim subst As String = "e"
         lbt(1) = subst

           Console.WriteLine("After editing strings...")
         For i = 0 To lbt.Count - 1
                Console.WriteLine("lbt({0}): {1}", i, lbt(i))
        Next
    End Sub
End Class
```

Figure 6.4: Output of the Code Above

```
After creation...
lbt(0): Hello
lbt(1): World
After adding strings...
lbt(0): Hello
lbt(1): World
lbt(2): W
lbt(3): I
lbt(4): J
lbt(5): t
After editing strings...
lbt(0): Hello
lbt(1): e
lbt(2): W
lbt(3): I
lbt(4): J
lbt(5): t
```

Source: http://www.java2s.com/Tutorial/VB/0120__Class-Module/DefineIndexerforyourownclass.htm

Self Assessment

Notes

Fill in the blanks:

10. ..... are nothing but natural extension of data fields.
11. An ..... allows you to access a class instance in terms of a member array.
12. For an explicit interface member implementation, the type is followed by an interface type, a period (.), and the keyword ..... .

6.4 Using Inheritance in Classes

One of the key features of Object-oriented Programming (OOP) languages is *inheritance*. Inheritance is the ability to use all of the functionality of an existing class, and extend those capabilities without re-writing the original class. Prior to the availability of Microsoft Visual Basic .NET, Visual Basic programmers did not have this capability. In Visual Basic .NET, you are able to inherit from classes that ship in the Microsoft .NET Framework, as well as from classes that you create. In this document, you will learn how to use inheritance, and see how it can significantly cut down your programming time.

A new class that is created by inheritance is sometimes called a *child class* or a *subclass*. The class you originally inherited from is called the *base class*, *parent class,* or the *superclass*. In some OOP languages, a base class may inherit from more than one base class. This means that if you had a Person class and a Car class, a Driver class might inherit all of the properties and methods from each of these two classes. In the .NET world, only single inheritance is allowed, so each subclass will have only one base class.

There are three types of inheritance that .NET supports: implementation, interface, and visual. Implementation inheritance refers to the ability to use a base class's properties and methods with no additional coding. Interface inheritance refers to the ability to use just the names of the properties and methods, but the child class must provide the implementation. Visual inheritance refers to the ability for a child form (class) to use the base forms (class) visual representation as well as the implemented code.

A class in .NET may inherit from a class that has already inherited from another class. In addition you may use an interface or even multiple interfaces within a class.

Inheritance is desirable because you want to avoid writing the same code over and over again. If you have two separate classes, and each one has to implement a **FirstName** and **LastName** property, you are going to have duplicate code. If you wish to change the implementation of one of these properties, you need to find all classes that have implemented these properties to make the changes. Not only is this time-consuming, but you also increase the risk of introducing bugs in the various classes. One thing to remember when you are considering using inheritance is that the relationship between the two classes should be an "is a" type of relationship. For example, an Employee is a Person, and a Manager is a Person, so these two classes can inherit from the Person class. But you should not have a Leg class inherit from a Person class as a Leg is not a person.

6.4.1 Importing in Class

Inheritance in VB.net is method by which the properties of the base classes are added to the derived classes. In Vb.net the keyword Inherits is used in the derived class to specify its base class. The MustInherit keyword is used to specify that class can be used only as a base class, NotInheritable is used to specify that a class cannot be inherited.

Notes

Example:

```
Module Module1
 Public Class s1
      Public a As Integer = 5
       Public Function val() As Integer
          Return a
      End Function
 End Class

 Public Class s2
      Inherits s1
      Public c As Integer = 20
       Public Function add() As Integer
          Return c + a
      End Function
 End Class

 Sub Main()
      Dim res As New s2
        System.Console.WriteLine("Final Value is::")
        System.Console.WriteLine(res.add())
       Console.Read()
 End Sub

End Module

Output:

Final Value is: 25
```

In the above example the value of **a** is inherited to from base class **s1** to the derived class **s2**. Using the instance **res** of class **s2** the values of 'a' as well as 'c' is added to give the result.

Self Assessment

True or False:

13. There are three types of inheritance that .NET supports: implementation, interface, and visual.
14. A new class that is created by inheritance is called a superclass.
15. The class you originally inherited from is called the subclass.

6.5 Summary

- Abstraction manages the complexities of a business problem by allowing you to identify a set of objects involved with that business problem.
- Encapsulation hides the internal implementation of an abstraction within the particular object.
- Polymorphism provides for multiple implementations of the same method.

Notes

- Inheritance allows you to reuse the interface of a class, but not its implementation.
- A Method is code that actually does something, that performs an action.
- In order to use a class, you first need to create an object from the class.
- Get procedure retrieves the property value from the class and the Set procedure assigns the property.
- Visual Basic .NET introduces true constructors that are executed whenever a new instance of the class is created.
- Constructors with parameters are called parameterized constructors.
- Shared constructors are used to initialize the shared variables of a type.
- Instance constructors are used to initialize variables that are declared with Dim, Public, Private, Friend, Protected, and Protected Friend keywords.
- An indexer allows you to access a class instance in terms of a member array.
- There are three types of inheritance that .NET supports: implementation, interface, and visual

6.6 Keywords

Abstraction: It manages the complexities of a business problem by allowing you to identify a set of objects involved with that business problem.

Encapsulation: It hides the internal implementation of an abstraction within the particular object.

Indexer: It allows you to access a class instance in terms of a member array.

Inheritance: It allows you to reuse the interface of a class, but not its implementation.

Instance constructors: They are used to initialize variables that are declared with Dim, Public, Private, Friend, Protected, and Protected Friend keywords.

Method: It is code that actually does something, that performs an action.

Polymorphism: It provides for multiple implementations of the same method.

Shared constructors: They are used to initialize the shared variables of a type.

6.7 Review Questions

1. What is object-oriented programming?
2. Give the steps to create a class.
3. Explain the methods in VB.NET.
4. What are constructors?
5. Differentiate between shared and instance constructors.
6. Explain the concept of properties in VB.Net.
7. What are the major differences between indexer and properties?
8. What is inheritance?

Notes

9. Explain the method to inherit a child class from a base class.
10. Does VB.NET support multiple inheritance?

Answers: Self Assessment

1.	Abstraction	2.	Encapsulation
3.	Polymorphism	4.	Inheritance
5.	Objects	6.	True
7.	True	8.	False
9.	False	10.	Properties
11.	Indexer	12.	This
13.	True	14.	False
15.	False		

6.8 Further Readings

Books

Beginning Vb.Net 2003, Willis.

Building Distributed Applications with Visual Basic.NET, Dan Fox, Sams.

Object-oriented Programming with Visual Basic.Net, Hamilton.

Visual Basic.NET Black Book, Steven Holzner.

Online links

http://msdn.microsoft.com/en-us/library/ms973814.aspx

http://www.hscripts.com/tutorials/vbnet/inheritance.html http://www.vkinfotek.com/oops/inheritance-visual-basic-net.html http://www.homeandlearn.co.uk/net/nets1p1.html

Notes

Unit 7: Namespaces

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the meaning and working of Namespaces
- Describe the object and class

Notes

- Understand the creating your first class library
- Define namespaces

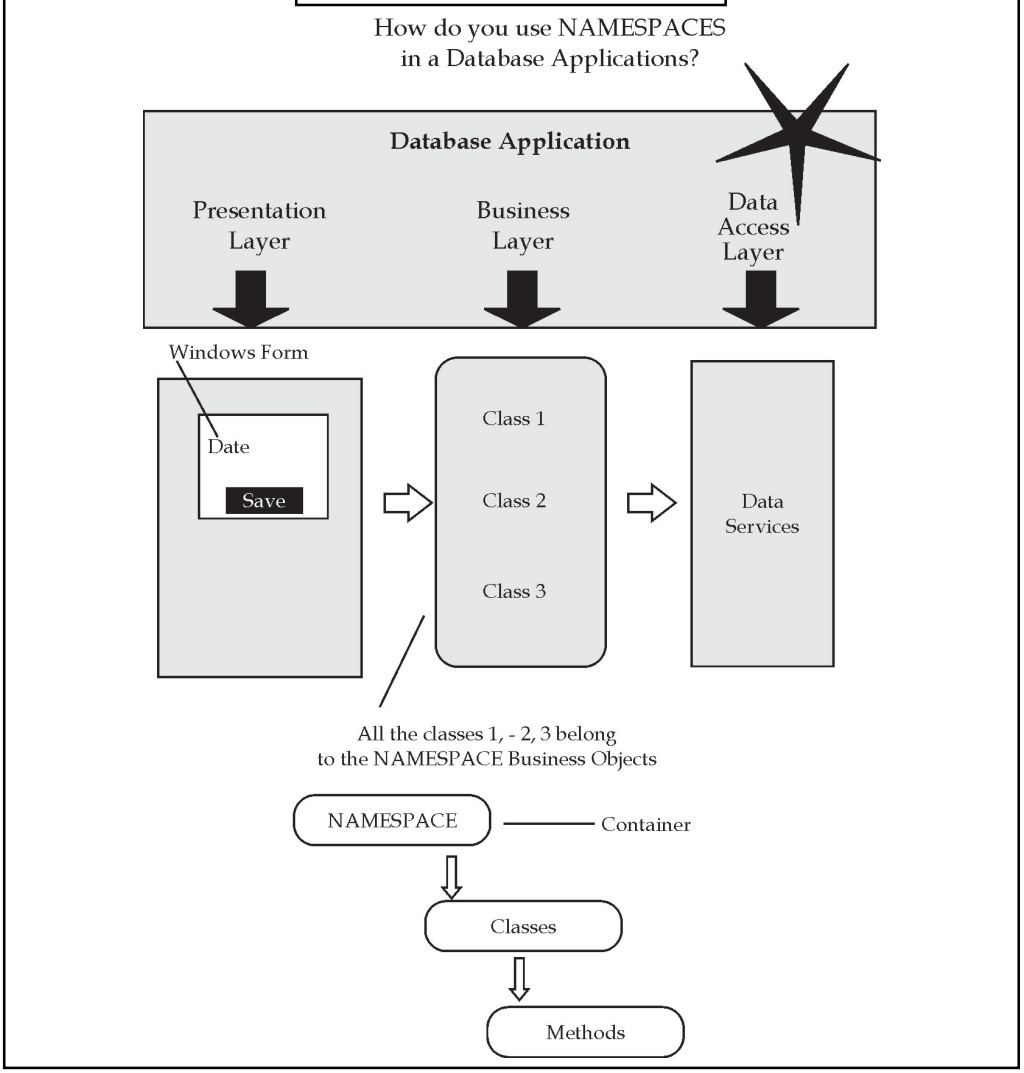
Introduction

Namespaces help you to create logical groups of related classes and interfaces. Namespaces allow us to organize Classes so that they can be easily accessed in other applications. Namespaces enable us to avoid any naming conflicts between classes that have the same name. We can use two classes with the same name in an application provided they belong to different namespaces.

7.1 Namespaces – Meaning and Its Working

Namespace is logical division of class, structure and interface or way to organize your Visual Basic .NET code is through the use of namespaces. They are a way of grouping type names and reducing the chance of name collisions. The namespace with all the built-in functionality comes under **System** namespace. All other namespaces comes under this **System** namespace.

Figure 7.1: Concept of Namespaces



Source: http://www.vkinfotek.com/namespace.html

Notes

Example: We create a new class named TextBox to accept only an integer type of value. However, if we want to use the VB.Net TextBox control in the same project, there will be a conflict in the name. When we create a Namespace, we will be able to easily distinguish between the Class TextBox created by us and the one provided by VB.Net.

Every project in Visual Basic.Net has a root namespace, which is set in the Property page of the project. When we create a project, by default, the name of the root namespace for the project is set to the name of the new project. For example, the root namespace for a project named MyProject is MyProject. Usually, when we create a project, we would like to give it a name which we like. So, we need to change the MyProject to the name which we wish. We will name the project as Acct1. We can also organize classes using the Namespace keyword.

Example: You can create a user-defined namespace called EmpNamespace in the project called Acct1 and place the class EmpClass within the EmpNamespace block.

```
Namespace EmpNamespace
Class EmpClass
End Class
End Namespace
The fully qualified or proper usage of the class EmpClass will be:
Acct1.EmpNamespace.EmpClass
The .Net framework uses a dot(.) as a delimiter between classes and
namespaces.
```

After we have created a Namespace, we will use the Namespace explicitly through direct addressing or implicitly through the Imports statement. Direct addressing involves directly accessing any class in the namespace by providing the fully qualified name.

Example of using fully qualified name is given below:

```
Microsoft.VisualBasic.MsgBox("Using Fully Qualified Name")
```

If we want to make all the classes in a given namespace available without the need to type the entire namespace each time, you can use the Imports statement. An example of using the Imports statement is given below:

```
Imports Microsoft.VisualBasic

...

MsgBox("Using Imports Statement")

Similarly, in the example we have chosen:

Imports Acct1.EmpNameSpace

Namespaces are of two types, Local and Global. Local Namespaces are
accessible only to the applications to which they belong. Global
Namespaces are accessible from all applications.
```

7.1.1 Some Namespaces and Their Use

We will just name a few System namespaces and discuss their basic function.

Notes

System: The System namespace contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.

System.Data: The System.Data namespace provides access to classes that represent the ADO.NET architecture. ADO.NET lets you build components that efficiently manage data from multiple data sources. In a disconnected scenario such as the Internet, ADO.NET provides the tools to request, update, and reconcile data in multiple tier systems. The ADO.NET architecture is also implemented in client applications, such as Windows Forms, or HTML pages created by ASP.NET.

***System.Collections*:** The System.Collections namespace contains interfaces and classes that define various collections of objects, such as lists, queues, bit arrays, hashtables and dictionaries.

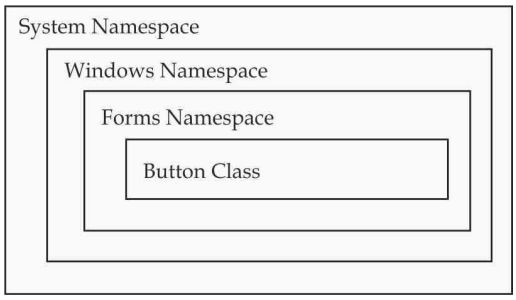
System.Drawing: The System.Drawing namespace provides access to GDI+ basic graphics functionality. More advanced functionality is provided in the System.Drawing.Drawing2D, System.Drawing.Imaging, and System.Drawing.Text namespaces. The Graphics class provides methods for drawing to the display device. Classes such as Rectangle and Point encapsulate GDI+ primitives. The Pen class is used to draw lines and curves, while classes derived from the abstract class Brush are used to fill the interiors of shapes.

System.Globalization: The System.Globalization namespace contains classes that define culture-related information, including language, country/region, calendars in use, format patterns for dates, currency, and numbers, and sort order for strings. These classes are useful for writing globalized (internationalized) applications. Classes such as StringInfo and TextInfo provide advanced globalization functionalities, including surrogate support and text element processing.

System.IO *:* The System.IO namespaces contain types that support input and output, including the ability to read and write data to streams either synchronously or asynchronously, to compress data in streams, to create and use isolated stores, to map files to an application's logical address space, to store multiple data objects in a single container, to communicate using anonymous or named pipes, to implement custom logging, and to handle the flow of data to and from serial ports.

System.Web: The System.Web namespaces contain types that enable browser/server communication. Child namespaces include types that support ASP.NET forms authentication, application services, data caching on the server, ASP.NET application configuration, dynamic data, HTTP handlers, JSON serialization, incorporating AJAX functionality into ASP.NET, ASP.NET security, and web services.

Figure 7.2: Namespace Hierarchy



Source: http://msdn.microsoft.com/en-us/library/ms973231.aspx

7.1.2 Declaring a Namespace

In VB.NET we declare a namespace as shown below:

- imports system;
- imports system.Data;

Notes

Example:

```
namespace ExampleNamespace
{
class TestExample
 {
public void ShowMessage()
{
Console.WriteLine("This is the TestExample namespace!");
}
  }
}
```

Using namespaces, we can establish security, version, reference, and deployment boundaries by using namespaces Because of grouping of namespaces we can create hierarchy which is easy to identify classes by fully qualified names.

7.1.3 Accessing Members of a Namespace

To access members of a namespace use the dot(.) operator. We can access variable, procedure, classes of a Namespace.

Example:

Let us explain the process to access a member using an example.

```
Imports Microsoft.VisualBasic
Namespace Namespace1
      Public Class Class1
            Public Sub MsgBox2()
                    MsgBox("You have accessed the Namespace correctly.")
            End Sub
      End Class
End Namespace
```

In order to access the sub we will need to import the namespace. Now, we will declare the Class variable as class Class1 inside the Page_Load sub. Then we will call the MsgBox2 sub so that the user will be sent a message box when the page is first loaded. Below is an example.

```
Imports Namespace1
Partial Class _Default Inherits System.Web.UI.Page
       Protected Sub Page_Load(ByVal Src As Object, ByVal E As EventArgs)
Handles MyBase.Load
            Dim theClass = New Class1
            theClass.MsgBox2()
      End Sub
End Class
```

Namespaces can hold several different elements such as classes, structures, interfaces, enumerations, and other namespaces. These will all be referenced with the same format as directed above.

7.1.4 Nesting a Namespace

You can declare one namespace within another. There is no strict limit to the levels of nesting you can declare, but remember that when other code accesses the elements declared in the

Notes

innermost namespace, it must use a qualification string that contains all the namespace names in the nesting hierarchy. All namespace names in your project are based on a root namespace. Visual Studio assigns your project name as the default root namespace for all code in your project.

For example, if your project is named Payroll, its programming elements belong to namespace Payroll. If you declare Namespace funding, the full name of that namespace is Payroll.funding.

If you want to specify an existing namespace in a Namespace statement, such as in the generic list class example, you can set your root namespace to a null value. To do this, click Project Properties from the Project menu and then clear the Root namespace entry so that the box is empty. If you did not do this in the generic list class example, the Visual Basic compiler would take System.Collections.Generic as a new namespace within project Payroll, with the full name of Payroll.System.Collections.Generic.

Example:

The following example declares two namespaces, one nested in the other.

```
Namespace n1
     Namespace n2
         Class a
               ' Insert class definition.
         End Class
     End Namespace
End Namespace
```

The following example declares multiple nested namespaces on a single line, and it is equivalent to the previous example:

```
Namespace n1.n2
     Class a
          ' Insert class definition.
     End Class
End Namespace
```

The following example accesses the class defined in the previous examples:

```
Dim instance As New n1.n2.a
```

The following example defines the skeleton of a new generic list class and adds it to the System.Collections.Generic namespace:

```
Namespace System.Collections.Generic
      Class specialSortedList(Of T)
          Inherits List(Of T)
           ' Insert code to define the special generic list class.
     End Class
End Namespace
```

7.1.5 Aliases of the Namespaces

Aliases enables type names to be referenced without namespace qualification. Its syntax is,

```
Imports [ aliasname = ] namespace
```

-or-

```
Imports [ aliasname = ] namespace.element
```

Notes

Where,

Aliasname: It is an optional parameter. An import alias or name by which code can refer to namespace instead of the full qualification string.

Namespace: It is a required parameter. The fully qualified name of the namespace being imported. Can be a string of namespaces nested to any level.

Element: It is an optional parameter. The name of a programming element declared in the namespace. Can be any container element.

An import alias defines the alias for a namespace or type. Import aliases are useful when you need to use items with the same name that are declared in one or more namespaces. You should not declare a member at module level with the same name as aliasname. If you do, the Visual Basic compiler uses aliasname only for the declared member and no longer recognizes it as an import alias. Although the syntax used for declaring an import alias is like that used for importing an XML namespace prefix, the results are different. An import alias can be used as an expression in your code, whereas an XML namespace prefix can be used only in XML literals or XML axis properties as the prefix for a qualified element or attribute name. If you supply element, it must represent a container element, that is, a programming element that can contain other elements. Container elements include classes, structures, modules, interfaces, and enumerations. The scope of the elements made available by an Imports statement depends on whether you specify element. If you specify only namespace, all uniquely named members of that namespace, and members of container elements within that namespace, are available without qualification. If you specify both namespace and element, only the members of that element are available without qualification.

Example: The following example includes Imports statements that create aliases for the referenced types. Aliases are used to specify the types.

```
Imports strbld = System.Text.StringBuilder
Imports dirinf = System.IO.DirectoryInfo

Public Function GetFolders() As String
     Dim sb As New strbld

      Dim dInfo As New dirinf("c:\")
      For Each dir As dirinf In dInfo.GetDirectories()
           sb.Append(dir.Name)
            sb.Append(ControlChars.CrLf)
     Next

      Return sb.ToString
End Function
```

7.1.6 Assemblies

An assembly is a collection of types and resources that forms a logical unit of functionality. All types in the .NET Framework must exist in assemblies; the common language runtime does not support types outside of assemblies. Each time you create a Microsoft Windows Application, Windows Service, Class Library, or other application with Visual Basic .NET, you're building a single assembly. Each assembly is stored as an .exe or .dll file.

Notes

Note Although it's technically possible to create assemblies that span multiple files, you're not likely to use this technology in most situations.

The .NET Framework uses assemblies as the fundamental unit for several purposes:

- Security
- Type Identity
- Reference Scope
- Versioning
- Deployment

Every assembly contains an assembly manifest, a set of metadata with information about the assembly. The assembly manifest contains these items:

- The assembly name and version
- The culture or language the assembly supports (not required in all assemblies)
- The public key for any strong name assigned to the assembly (not required in all assemblies)
- A list of files in the assembly with hash information
- Information on exported types
- Information on referenced assemblies

In addition, you can add other information to the manifest by using assembly attributes. Assembly attributes are declared inside of a file in an assembly, and are text strings that describe the assembly.

Example: You can set a friendly name for an assembly with the AssemblyTitle attribute:

```
<Assembly: AssemblyTitle("Test Project")>
```

Self Assessment

Fill in the blanks:

1. ..... allow us to organize Classes so that they can be easily accessed in other applications.
2. ..... addressing involves directly accessing any class in the namespace by providing the fully qualified name.
3. ..... Namespaces are accessible only to the applications to which they belong.
4. ..... Namespaces are accessible from all applications.
5. All other namespaces comes under this ..... namespace.
6. To access members of a namespace use the ..... operator.
7. ..... enables type names to be referenced without namespace qualification.
8. An ..... is a collection of types and resources that forms a logical unit of functionality.

Notes

7.2 Object and Class

We will now discuss a little about the basic concepts used in VB.NET.

Classes and Objects

In VB.NET, a class is a collection of properties and methods. If you look right at the top of the code window for a Form, you'll see:

Public Class Form1

The word "Public" means that other code can see it. Form1 is the name of the Class

If you look at the bottom of the coding window, you'll see End Class, signifying the end of the code for the Class. When you place a Button or a textbox on the Form, you're really adding it to the Form Class. When you start the Form, VB does something called instantiation. This basically means that your Form is being turned into an Object, and all the things needed for the creation of the Form are being set up for you (Your controls are being added, variables are being set up an initialised, etc.).

And that's the basic difference between a Class and an Object: A Class is the code itself; the code becomes an Object when you start using it.

Properties

In VB.NET, properties are nothing but natural extension of data fields. They are usually known as 'smart fields' in VB.NET community. We know that data encapsulation and hiding are the two fundamental characteristics of any object oriented programming language. In VB.NET, data encapsulation is possible through either classes or structures. By using various access modifiers like private, public, protected, internal, etc. it is possible to control the accessibility of the class members. Usually inside a class, we declare a data field as private and will provide a set of public SET and GET methods to access the data fields. This is a good programming practice, since the data fields are not directly accessible outside the class. We must use the set/get methods to access the data fields.

Methods

A method created in a Class is nothing more than a Function or a Sub. A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. It operates on any object of the class of which it is a member, and has access to all the members of a class for that object. Member variables are attributes of an object (from design perspective) and they are kept private to implement encapsulation. These variables can only be accessed using the public member functions.

Events

An event is a message sent by an object announcing that something has happened. Events are implemented using delegates, a form of object-oriented function pointer that allows a function to be invoked indirectly by way of a reference to the function.

Notes

Self Assessment

True or False:

9. A member function of a class is a function that has its definition or its prototype outside the class definition.

10. Member variables are attributes of an object and they are kept private to implement encapsulation.

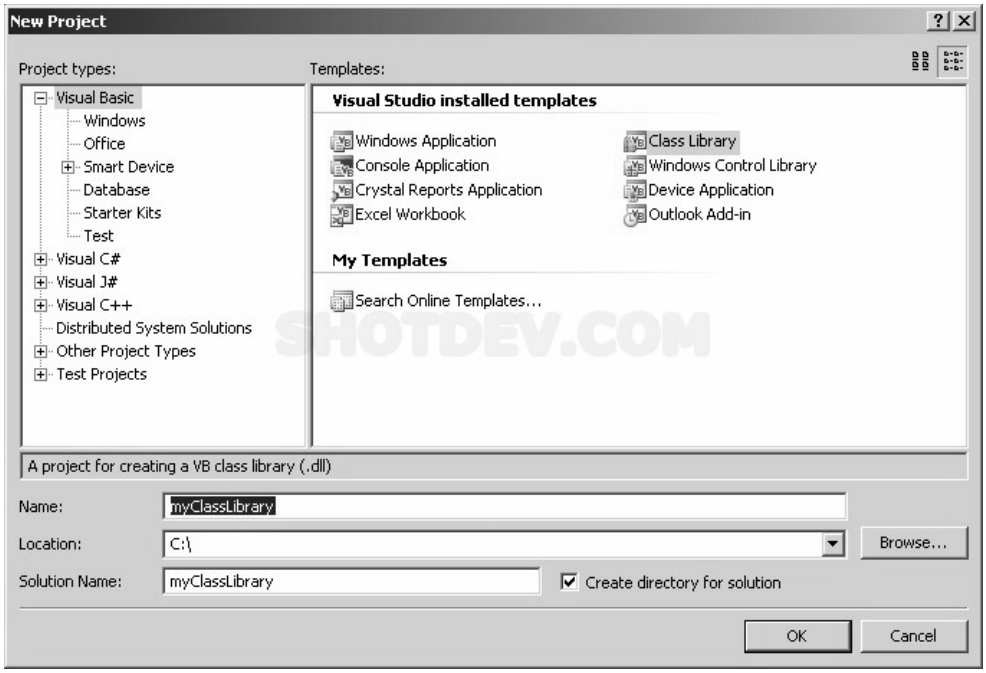
12. An action is a message sent by an object announcing that something has happened.

7.3 Creating Your First Class Library

Let us see the process to create a class library.

Step 1: Create a new project but instead of selecting **Windows Forms Application**, you want to select **Class Library** as shown below.

Figure 7.3: Creating a Class Library



Source: http://www.shotdev.com/aspnet/vbnet-component/vbnet-create-class-library-dll/

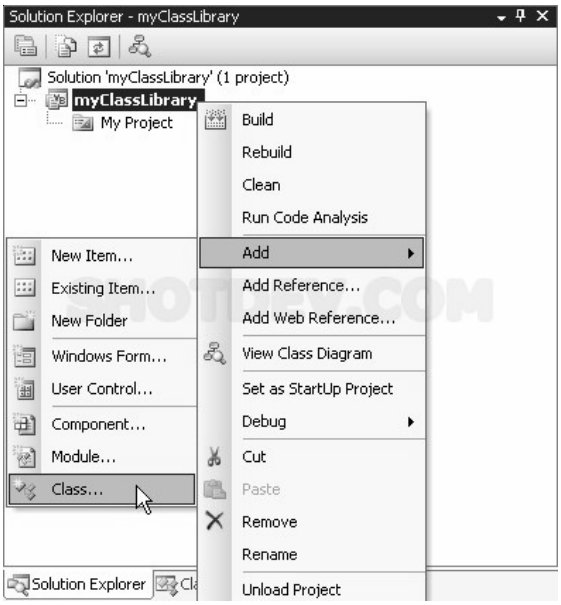
Step 2: You will be presented with

```
Public Class Class1

End Class
```

Let us change the class name to **MyFunctions**.

Notes

Figure 7.4: Adding to a Class Library



Source: http://www.shotdev.com/aspnet/vbnet-component/vbnet-create-class-library-dll/

Step 3: We will program a simple Math function, which will use 2 textboxes that I will have in my application, and add the 2 numbers together and then display the result in a label.

```
Public Class MyFunctions
   Public Function AddMyValues(ByVal Value1 As Double, ByVal Value2 As
Double)
       Dim Result As Double

       Result = Value1 + Value2

       Return Result

   End Function
End Class
```

We created a Function called **AddMyValues** with **ByVal Value1 as Double** which will be the holder for the value being passed from textbox 1, and the same for **Value2** which would hold the value being passed from textbox 2.

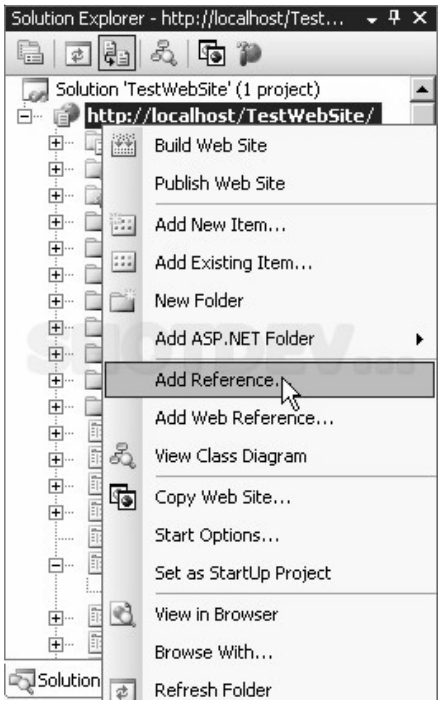
Step 4: Save the project, and then build it just like for an application. Now go to the projects Bin/Debug directory and you will find a DLL that you have just created.

Step 5: Now let us create an application to use it. Create a new application like you would normally do and create a form similar to the one shown. Now, we will leave the default names for the textboxes, but we have changed the name of the Label to **lblResult** and the name of the button to **btnAdd**.

Step 6: Now you will have to add a reference to your newly created DLL, To do that, choose Project -> Add Reference, and browse to where your DLL is located, select it and click ok.

Notes

Figure 7.5: Adding Reference



Source: http://www.shotdev.com/aspnet/vbnet-component/vbnet-create-class-library-dll/

Step 7: Now let us do some coding. First you need to import it as you would with others. So, above Public Class Form1 type **Imports** and a list should show up, and select your DLL. Imports Abc

Step 8: Now in the Button_Click event, we added this line Dim Add As New Abc.MyFunctions. A name **Add** is declared for **MyFunctions** in DLL. Thus, we don't have to type **Abc.MyFunctions.AddMyValues**, instead we can just use **Add.AddMyValues**.

Now we will use that variable to call the required function on the DLL that I want to use. The whole code is shown below. Study it and you will see how **Add** is used. Also look back at the DLL code at the start to fully understand it.

```
Imports Abc
Public Class Form1

   Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnAdd.Click
      Dim Add As New Abc.MyFunctions

      lblResult.Text = Add.AddMyValues(CDbl(TextBox1.Text),
CDbl(TextBox2.Text)).ToString

   End Sub
End Class
```

Step 9: Now the code is done, save the project and then run the application. Enter 2 numbers and then press the button and the result – which was done in the DLL will be displayed in your result label.

Notes

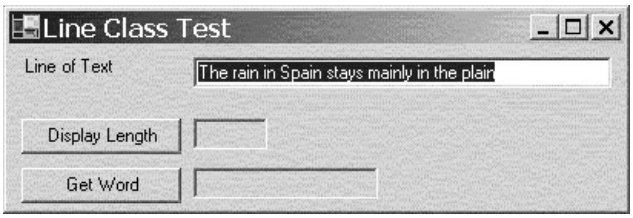
7.3.1 Adding a "Souped-Up" Class

Classes are the heart and soul of an object-oriented language. You will find yourself using classes whenever you write even the simplest of programs in Visual Basic .NET. The Microsoft .NET Framework makes extensive use of classes, and so should you. Below are some common uses of classes:

- Wrapping up the representation and set of operations you perform on a database table, for example adding, editing, deleting, and retrieving data.
- Wrapping up the set of operations and data for dealing with text files such as reading, writing, and indexing the lines of text within the file.
- Wrapping up all global variables in a program into properties within a class. This can help with keeping track of the amount of "free-floating" globals that somehow seem to work their way into many programs.

Let's create a class representing a line of text. To do this, you create a property to return the line of text and a read-only property that returns the length of the text. You will also create a method that returns the first word in the line. As you perform all of these steps, you will learn the correct way to create a class. You will build a form like the one shown below to test the Line class as you build it.

Figure 7.6: Adding a Class



Source: http://msdn.microsoft.com/en-us/library/ms973814.aspx

To add the class follow the steps below:

1. Open the Add New Item dialog box by clicking Project and then clicking Add Class.
2. Set the Name property to Line and click OK.
3. You will now see a new file appear in your project and a code window within the Visual Studio .NET environment. In the Code window, there will be some code that looks like this:

```
Public Class Line

    End Class
```

All of the properties and methods that you create for this class must be entered between these lines of code.

7.3.2 Creating Properties

To create a property within a class, you can either create a field (i.e.. a Public variable), or you can create a Private variable and expose the Private variable using a Property statement. There are several reasons why you want to only expose properties through a Property statement.

- You can create a read-only or write-only property, as opposed to a Public variable, which will always be read-write.

Notes

- You can add error handling within a Property statement to check for invalid values being set. You can't check for invalid values when setting a Public variable because there is no code that runs in response to setting a Public variable.
- You can expose calculated values as properties even if they are not stored as actual data within the class. An example of this is a Length property. You probably don't want to store the length of a line of text, as it could change.

You will now create two properties named Line and Length for the Line class. You will first create a private variable to hold the line of data that you store within the class. Next, you will create the Property statements for these two new properties. Modify the class in your project so it looks like the code shown below:

```
Public Class Line

     Private mstrLine As String

     Property Line() As String
        Get
            Return mstrLine
        End Get
         Set(ByVal Value As String)
             mstrLine = Value
        End Set
    End Property

     ReadOnly Property Length() As Integer
       Get
             Return mstrLine.Length
       End Get
    End Property
End Class
```

7.3.3 Building a Test Client

We now need to test this class. We need to see how to call this class in a client application. From the File menu, choose New and then Project. This time, add a Windows Application. After you click the OK button, the new project is loaded into the Solution Explorer.

7.3.4 Read-only and Write-only Properties

In the code above, you also created a read only property by using the ReadOnly keyword in front of the Property statement. When you use this keyword, Visual Studio .NET adds a Get…End Get block to the Property statement. In fact, if you try to add a Set…End Set block, you will receive a compiler error. If you wanted to create a read-only property in Visual Basic 6.0, you did not create the Property Let procedure, but no error was generated if you left it off. You just were unable to set the property at run time. In the project you have created, you have one class and one form. You will now write code in the form that creates a new Line object, places a line of text into the Line property of your object, and then places the length of the line into the Text property of the txtLength text box on your form.

1. In the Solution Explorer window, double-click the frmLineTest form to bring up the form in design mode.

Notes

2. Double-click the Display Length button. Visual Basic .NET creates a btnDisplay_Click event procedure for you in the code behind this form. All you need to do is fill in the lines of code shown below, in the body of the procedure.

```
Private Sub btnDisplay_Click( _
   ByVal sender As Object, _
   ByVal e As System.EventArgs) Handles btnDisplay.Click

     Dim oLine As Line

     oLine = New Line()

      oLine.Line = txtLine.Text
      txtLength.Text = oLine.Length.ToString()
End Sub
```

Within this event procedure, you create a variable named oLine by using the Dim statement. This variable is defined as a reference to a Line class. You actually create the new object reference by using the New keyword, as shown in the next line after the Dim statement. The New keyword must be followed by the name of the class you wish to instantiate. Another difference between Visual Basic .NET and Visual Basic 6.0 is that you no longer use the Set keyword when creating a new object.

In Visual Basic .NET, you are allowed to combine these two lines into one, as shown in the code below:

```
Dim oLine As Line = New Line()
```

Or

```
Dim oLine as New Line()
```

Visual Basic .NET (and all .NET languages) allow you to declare and initialize any variable on the same line. In Visual Basic 6.0, you were unable to do this, as the Dim statement was not an executable line of code. In Visual Basic .NET, Dim is an executable line of code so this syntax is perfectly legal.

Now let's examine the next two lines of this event procedure:

```
oLine.Line = txtLine.Text
txtLength.Text = CStr(oLine.Length)
```

The first line sets the Line property in your object to be equal to the value contained in the Text property of the txtLine text box on the form. This passes the data in the Text property to the Value variable in the Set…End Set block in the Line object.

Now you are ready to report back the length of the string contained in the txtLine text box. Remember, you set the value of this text box equal to the string "The rain in Spain stays mainly in the plain." This is the value that is contained in the mstrLine variable within your object. You can invoke the Length property on your Line object and it will return the length of this particular string. Because the Length property is an Integer value, you need to convert that value to a string before you can place it into the Text property of the txtLength text box. You accomplish this by applying the ToString method to the Length property.

A write-only property is one that defines a Set…End Set block and no Get…End Get block. You are allowed to place data into this property only at run time, but are unable to retrieve data from this property. If you wanted to extend this Line class to be able to read the line of text in from a file on disk, you might pass the file name to this class. You could accomplish this by using a write-only property. Here is an example of what a write-only property might look like.

Notes

```
WriteOnly Property FileName() As String
      Set(ByVal Value As String)
          mstrFileName = Value
     End Set
End Property
```

This syntax is again different from Visual Basic 6.0 in that you use the WriteOnly keyword as a prefix to the Property statement. Visual Studio .NET creates the Set…End Set block for you automatically. If you try to add a Get…End Get block, the compiler will give you an error.

7.3.5 Parameterized Properties

It is also called as Property arrays. Since properties with parameters can also be used to store multiple values. If you want to have a look or feel the multiple properties, here is the example, consider the HashTable which contains the Item property. If you can observe it very closely you get a clear picture of what is parameterised property and how it can be used.

7.3.6 Default Properties

A property that accepts arguments can be declared as the default property for a class. A *default property* is the property that Microsoft Visual Basic .NET will use when no specific property has been named for an object. Default properties are useful because they allow you to make your source code more compact by omitting frequently used property names.

The best candidates for default properties are those properties that accept parameters and that you think will be used the most often. For example, the **Item** property is a good choice for the default property of a collection class because it is used frequently.

The following rules apply to default properties:

- A type can have only one default property, including properties inherited from a base class. There is one exception to this rule. A default property defined in a base class can be shadowed by another default property in a derived class.
- If a default property from a base class is shadowed by a non-default property in a derived class, the default property is still accessible using default property syntax.
- A default property may not be **Shared** or **Private**.
- If an overloaded property is a default property, all overloaded properties with that same name must also specify **Default**.
- Default properties must accept at least one argument.

7.3.7 Constructors in Your Classes

Constructors used in a class are member functions to initialize or set the objects of a class in VB.net. They don't return any value and are defined in a Sub with a keyword New. Multiple constructors can be created in class with any access specifies, by default constructors are of Public access type.

Example:

```
Public Class Sample
 Private a As Integer
   Public Sub New(ByVal setval As Integer)
```

Notes

```
            a = setval
        End Sub

         Public Function disp()
            Return a
        End Function
    End Class

    Sub Main()
         Dim d As New Sample(5)
           Console.WriteLine("Value of a is initialized to:")
           Console.WriteLine(d.disp())
         Console.Read()
    End Sub

Ouput:
```

Value of a is initialized to:

5

In the above example, using the Constructor declared within the sub procedure **New** the value of a is set to 5.

7.3.8 Classes without Constructors

To create a new class we use the Class...End Class block. Whether or not a class has constructors or implements System.ComponentModel.Component, the class can still have properties and methods, and can be created just like any other class.

7.3.9 Adding Methods to Classes

A method in a class can be a procedure that performs some sort of operation on the data within the class. Or a method could be a function that performs some operation on the data, and returns that data back from the class. To be able to call a method from an instance of this class, the method must be declared Public. If a method is declared Private, only other methods within the class can call that method. Creating a method in Visual Basic .NET is exactly the same as in Visual Basic 6.0. Let's create a method in the Line class that breaks up the line of text passed into the class into each separate word. This method is a function that returns the first word contained in the line of text you pass in. If you used the string that was given to you when creating the form, the line of text will be "The rain in Spain stays mainly in the plain." In this case, the first word returned will be "The."

Open the Line.vb class in design mode.

Just below your Property statements, create the following method:

```
Public Function GetWord() As String
     Dim astrWords() As String

     astrWords = Split(mstrLine, " ")

     Return astrWords(0)
End Function
```

Notes

7.3.10 Adding Events

.NET lets you add and remove Event Handlers dynamically on the fly. Your code can start and stop handling events at any time during program execution. Also, you can tie the same code (event handler) to multiple events similar to the Handles clause in VB.NET. The VB.NET AddHandler and RemoveHandler statements allow this behavior. Both statements take two arguments: the name of the event to handle and the name of the procedure that will handle the event.

```
Public Class MyClass
          Public Event MyEvent       //   Declare the event to be raised.

          Sub RaiseMyEvent()

               RaiseEvent MyEvent //   Method to raise the event.

         End Sub
     End Class

     Public Class Test
        Sub TestEvent()
            Dim myobj As New MyClass()  //   Create the object that raises
the event you want to handle.

              AddHandler myobj.myEvent, AddressOf MyEventHandler // Associate
the code to be executed (event handler) with the event.

 myObj.RaiseMyEvent()     // Tell the object to raise the event. It will be
handled by sub MyEventHandler.

RemoveHandler myobj.myEvent, AddressOf MyEventHandler  //  Dis-associate
the handler from the event.
            myObj.RaiseEvent() // Tell the object to raise the event. It will
NOT be handled by sub MyEventHandler.

      End Sub

        Sub MyEventHandler()
             MsgBox("I caught the event!")   // Code to be executed when the
event is raised.

       End Sub
     End Class
```

Self Assessment

Fill in the blanks:

12. Read only property by using the ..... keyword in front of the Property statement.
13. ..... used in a class are member functions to initialize or set the objects of a class.

Notes

7.4 Creation of Namespaces

Namespaces allow you to create a system to organize your code. A good way to organize your namespaces is via a hierarchical system. You put the more general names at the top of the hierarchy and get more specific as you go down. This hierarchical system can be represented by nested namespaces. By placing code in different sub-namespaces, you can keep your code organized.

7.4.1 Creating Your Own Namespaces

A *namespace* designates a collection of programming elements, organized and classed for grouping operations and easy access. At the namespace level, the programming elements include:

- Classes of objects [...];
- Structures [...];
- Modules [...];
- Interfaces [...];
- Delegates [...];
- Enumeration [...];
- Other namespaces.

Namespaces can be organized, aggregated into an assembly [...]. An assembly can contain multiple namespaces, which can contain other namespaces.

By default, the executable file created with Visual Studio environment contains a namespace with a name identical to that of project. More assemblies can use the same namespace.

Explicit definition of a namespace is done in VB.NET with specifications **Namespace** and **End Namespace**. It considers an application of type **Console Application**. It defines the namespace *ProiectNou* that contains:

- A namespace called *Formular1*;
- A class called *GrupOp* which contains two methods: *Adun* and *Imp*.

The namespace *Formular1* contains the class *GrupOp* with an event two methods: *Dif* and *Prod*. It notes the presence of two classes of objects with the same name *GrupOp*, but included in different namespaces: *ProiectNou*, respectively *Formular1. The* namespace *ProiectNou* is defined on the same level as the procedure **Main** is defined.

```
Namespace  ProiectNou

      Namespace  Formular1
          Class GrupOp
                Public Event CalculOp(ByVal x As Integer, ByVal y As Integer)

                Public Function Dif(ByVal a As Integer, ByVal b As Integer)_
As Integer
                  Return a - b
              End Function

                Public Function Prod(ByVal a As Integer, ByVal b As Integer)_
As Long
```

Notes

```
                    Return a * b
                End Function
            End Class
        End Namespace

        Class GrupOp
            Public Function Adun(ByVal a As Integer, ByVal b As Integer)_
As Integer
                Return a + b
            End Function

            Public Function Imp(ByVal a As Integer, ByVal b As Integer)_
As Double
                Return a / b
            End Function
        End Class

End Namespace

Module Module1

    Sub Main()
        Dim calc As NameSpaceApp.ProiectNou.GrupOp
        calc = New NameSpaceApp.ProiectNou.GrupOp
        MsgBox("4+6=" & calc.Adun(4, 6))
        Dim calcNou As NameSpaceApp.ProiectNou.Formular1.GrupOp
        calcNou = New NameSpaceApp.ProiectNou.Formular1.GrupOp
        MsgBox("4*6=" & calcNou.Prod(4, 6))
    End Sub

End Module
```

In the procedure **Main**, it is noted that to define a variable of type *GrupOp* it has to specify the complete path to the class *GrupOp* by expressions *NameSpaceApp.ProiectNou.GrupOp* and *NameSpaceApp.ProiectNou.Formular1.GrupOp*. Otherwise, the application cannot be compiled because it does not "know" where to extract the definition of the class *GrupOp* from. *NameSpaceApp* is namespace at project level created by Visual Studio environment. Referring of a programming item in VB.NET source code is done by specification **Imports**. Referring the programming elements is done starting with the namespace with the highest level of aggregation and it provides the paths of subcollections included in namespace by using operator.

```
Imports NameSpaceApp.ProiectNou.Formular1
```

Thus, it accesses the programming element *GrupOp* included in *Formular1*, which it is included in the namespace *ProiectNou*. The namespace *ProiectNou* is defined in the namepsace at the application level, called with the same name like the project *NameSpaceApp*. The complete referring of the list of items in the programming elements included in namespaces, avoids conflict by their given identical names.

For ease of operation with elements of namespace, it can assign an alias to refer a program element included in a namespace. Alias allows direct reference to this element.

```
Imports GrupOperatii = NameSpaceApp.ProiectNou.Formular
```

Notes

If the first example, namespace was defined in the same file with VB.NET source code module that has been used elements defined in *ProiectNou*. If ProiectNou namespace is defined in another source file included in the project *NameSpaceApp*, then before developing the module *Module1* it should be included specifications **Imports**, as it follows:

```
Imports G1 = NameSpaceApp.ProiectNou
Imports G2 = NameSpaceApp.ProiectNou.Formular1

Module Module1

     Sub Main()
          Dim calc As G1.GrupOp
          calc = New G1.GrupOp
           MsgBox("4+6=" & calc.Adun(4, 6))
          Dim calcNou As G2.GrupOp
          calcNou = New G2.GrupOp
           MsgBox("4*6=" & calcNou.Prod(4, 6))
     End Sub

End Module
```

The namespace *ProiectNou* has the same definition, but included in another source file in the application *NameSpaceApp*. In a namespace, the following programming elements cannot be defined: properties [...], procedures [...], variables [...] and events [...]. These elements are defined in source code containers: modules, structures and classes.

Case Study

Using the .NET Framework Queue Class in VB.NET

Thread Safety

Only the following methods are safe for multithreaded operations: BeginPeek, BeginReceive, EndPeek, EndReceive, GetAllMessages, Peek, and Receive.

Remarks

Message Queuing technology allows applications running at different times to communicate across heterogeneous networks and systems which might be temporarily offline. Applications send, receive, or peek (read without removing) messages from queues. Message Queuing is an optional component of Windows 2000 and Windows NT, and must be installed separately.

The MessageQueue class is a wrapper around Message Queuing. There are multiple versions of Message Queuing, and using the MessageQueue class can result in slightly different behavior, depending on the Operating System you are using. For information about specific features of each version of Message Queuing, see the topic "What's New in Message Queuing" in the Platform SDK in MSDN.

The MessageQueue class provides a reference to a Message Queuing queue. You can specify a path in the MessageQueue constructor to connect to an existing resource, or you can create a new queue on the server. Before you can call Send, Peek, or Receive, you must

Contd...

Notes

associate the new instance of the MessageQueue class with an existing queue. At that point, you can manipulate the queue properties such as Category and Label.

The MessageQueue class supports two types of message retrieval: synchronous and asynchronous. The synchronous methods, Peek and Receive, cause the process thread to wait a specified time interval for a new message to arrive in the queue. The asynchronous methods, BeginPeek and BeginReceive, allow the main application tasks to continue in a separate thread until a message arrives in the queue. These methods work by using callback objects and state objects to communicate information between threads.

When you create a new instance of the MessageQueue class, you are not creating a new Message Queuing queue. Instead, you can use the Create, Delete, and Purge methods to manage queues on the server.

Unlike Purge, Create and Delete are static (Shared in Visual Basic) members, so you can call them without creating a new instance of the MessageQueue class.

You can set the MessageQueue object's Path property with one of three names: the friendly name, the FormatName, or the Label. The friendly name, which is defined by the queue's MachineName and QueueName properties, is *MachineName\ QueueName* for a public queue, and *MachineName\ Private$\ QueueName* for a private queue. The FormatName property allows offline access to message queues. Lastly, you can use the queue's Label property to set the queue's Path.

For a list of initial property values for an instance of MessageQueue, see the MessageQueue constructor.

Using the Message Queue

Create an instance of the class MessageQueue. Before you can create it, you need to import a reference of the class:

Imports System.Messaging

'' Creating Instance of Message Queue

'' We have passed in the path Of the Queue

'' the path declared is a private queue path

Private WithEvents myQueue As New MessageQueue(".\private$\myQueue")

'' You can define the path later on also

Private WithEvents myQueue As New MessageQueue()

'' then in the form load or any other event define the path

'' the path declared is a private queue path

myQueue.Path = "".\private$\myQueue"

We have declared the message queue with an event because we are going to use some of the events available with the class.

Now, let's see how to send a message to the queue:

To send a message to the message queue, you can either create a message using the Message class or by directly sending a string:

```
' sending by creating a message variable
Dim msg As New Message
```

Contd...

Notes

```
' it is the label of the message
msg.Label = "Example"
' it is the body of the message
msg.Body = "This message is send as an example"
myQueue.Send(msg)

' Send msg with creating a message variable
myQueue.Send("Example","This message is send as an example")
```

There are some other ways to send messages to the queue. I found these methods over the MSDN. I found some of them very useful.

```
' References public queues.
Public Sub SendPublic()
          ' the path is of public queue
           Dim myQueue As New MessageQueue(".\myQueue")
            myQueue.Send("Public queue by path name.")
          Return
End Sub 'SendPublic

' References private queues.
Public Sub SendPrivate()
      Dim myQueue As New MessageQueue(".\Private$\myQueue")
            myQueue.Send("Private queue by path name.")
          Return
End Sub 'SendPrivate

' References queues by label.
      Public Sub SendByLabel()
      Dim myQueue As New MessageQueue("Label:TheLabel")
            myQueue.Send("Queue by label.")
     Return
End Sub 'SendByLabel

' References queues by format name.
Public Sub SendByFormatName()
     Dim myQueue As New _
             MessageQueue("FormatName:Public=5A5F7535-AE9A-41d4-935C-
845C2AFF7112")
      myQueue.Send("Queue by format name.")
     Return
End Sub 'SendByFormatName
```

Now, let's see how to receive message from the queue:

By using the method Receive, we can receive the first message in the queue, removing it from the queue:

```
' using the receive method
myQueue.Receive()
```

But there is a catch: we have to store the message we received in a Message variable.

```
' saving message from the queue in our temp message variable
Dim tempMsg as New Message
tempMsg = myQueue.Receive()
```

Contd...

Notes

If you wish to check the message before removing it, you can use the Peek method:

```
' using the peek method
Dim tempMSg as New Message
tempMsg = MyQueue.Peek()
' Check if the message is ours
If tempMsg.Label = "Example" Then
     ' we got the message
     ' so we remove it from the Queue
     myQueue.Receive()
End If
```

Points of Interest

The MessageQueue is useful when there are a couple of applications that share data between them.

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://www.codeproject.com/Articles/23883/Using-MessageQueue-in-the-NET-Framework-2-0

Self Assessment

True or False:

14. Namespaces allow you to create a system to organize your code.
15. An assembly cannot contain multiple namespaces.

7.5 Summary

- Namespaces help you to create logical groups of related classes and interfaces.
- Namespaces enable us to avoid any naming conflicts between classes that have the same name.
- Every project in Visual Basic.Net has a root namespace, which is set in the Property page of the project.
- Direct addressing involves directly accessing any class in the namespace by providing the fully qualified name.
- Local Namespaces are accessible only to the applications to which they belong.
- Global Namespaces are accessible from all applications.
- All other namespaces comes under the System namespace.
- To access members of a namespace use the dot(.) operator.
- Aliases enables type names to be referenced without namespace qualification.
- An assembly is a collection of types and resources that forms a logical unit of functionality.
- A member function of a class is a function that has its definition or its prototype within the class definition like any other variable.

Notes

- Member variables are attributes of an object (from design perspective) and they are kept private to implement encapsulation.
- Event is a message sent by an object announcing that something has happened.
- A property that accepts arguments can be declared as the default property for a class.
- Constructors used in a class are member functions to initialize or set the objects of a class.

7.6 Keywords

Aliases: They enables type names to be referenced without namespace qualification.

A***ssembly:*** It is a collection of types and resources that forms a logical unit of functionality.

Direct addressing: It involves directly accessing any class in the namespace by providing the fully qualified name.

Global Namespaces: They are accessible from all applications.

Local Namespaces: They are accessible only to the applications to which they belong.

Member function: It is a function that has its definition or its prototype within the class definition like any other variable.

Member variables: They are attributes of an object (from design perspective) and they are kept private to implement encapsulation.

Namespaces: They help you to create logical groups of related classes and interfaces.

7.7 Review Questions

1. What is a namespace?
2. Why do we need namespaces?
3. Can we create nested namespaces? If yes, how?
4. How to create an alias?
5. What are assemblies?
6. Why do we need objects?
7. Can we have default constructors?
8. Differentiate between read-only and write-only properties.
9. Give an example to show the process to create a class library.
10. What is an event?

Answers: Self Assessment

1.	Namespace	2.	Direct
3.	Local	4.	Global
5.	System	6.	Dot
7.	Alias	8.	Assembly
9.	False	10.	True

Notes

11. False
12. Constructors
13. ReadOnly
14. True
15. False

7.8 Further Readings

Books

Building Distributed Applications with Visual Basic.NET, Dan Fox, Sams.

Object-oriented Programming with Visual Basic.Net, Hamilton.

Programming Visual Basic .NET, J. Liberty.

VB .NET Language in a Nutshell, Steven Roman, Ron Petrusha, Paul Lomax.

Online links

http://www.shotdev.com/aspnet/vbnet-component/vbnet-create-class-library-dll/ http://pietschsoft.com/post/2007/11/26/NET-How-to-Alias-Namespaces-and-Data-Types.aspx

http://www.dotnetheaven.com/article/namespaces-and-assemblies-in-vb.net

http://msdn.microsoft.com/en-us/magazine/cc164108.aspx

http://www.dreamincode.net/forums/topic/121314-creating-a-dll-and-using-it-in-your-application/

Notes

Unit 8: Exception Handling

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the using try and catch blocks for exception handling
- Define the try finally section

Introduction

An exception is a problem that arises during the execution of a program. An exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. VB.Net exception handling is built upon four keywords: Try, Catch, Finally and Throw.

- ***Try:*** A Try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more Catch blocks.
- ***Catch:*** A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The Catch keyword indicates the catching of an exception.
- ***Finally:*** The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- ***Throw:*** A program throws an exception when a problem shows up. This is done using a Throw keyword.

Notes

Assuming a block will raise and exception, a method catches an exception using a combination of the Try and Catch keywords. A Try/Catch block is placed around the code that might generate an exception. Code within a Try/Catch block is referred to as protected code, and the syntax for using Try/Catch looks like the following:

```
Try
     [ tryStatements ]
     [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
     [ catchStatements ]
     [ Exit Try ] ]
[ Catch ... ]
[ Finally
     [ finallyStatements ] ]
End Try
```

You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.

8.1 Concept of Exception Handling

Only perfect programmers create perfect code from the beginning. The rest must address imperfections along the way to developing a successful application. Luckily for us MicrosoftVisual Basic .NET offers two ways of handling exceptions.

- Unstructured, follows the exception-handling conventions of earlier versions of Visual Basic.
- Structured, handles exceptions in ways that resemble exception handling in Microsoft Visual C# and Microsoft Visual C++.

The terms, error and exception, are often used interchangeably. In fact, an error, which is an event that happens during the execution of code, interrupts or disrupts the code's normal flow and creates an exception object. When an error interrupts the flow, the program tries to find an exception handler — a block of code that tells it how to react — that will help it resume the flow. In other words, an error is the event; an exception is the object that the event creates. Programmers use the phrase "throwing an exception" to mean that the method in question encountered an error and reacted by creating an exception object that contains information about the error and when/where it occurred. Factors that cause errors and subsequent exceptions include user error, resource failures, and failures of programming logic. Such errors are related to how the code undertakes a specific task; they are not related to the purpose of the task. Exception handling means interpreting and reacting to the exceptions created by errors.

Example: When your application asks the user to input a telephone number, the following assumptions come into play:

- The user will input a number rather than characters.
- The number will have a certain format.
- The user will not input a null string.
- The user has a single telephone number.

User input might violate any or all of these assumptions. Robust code requires adequate exception handling, which allows your application to recover gracefully from such a violation.

Notes

Unless you can guarantee that a method will never throw an exception under any circumstances, allow for informative exception handling. Exception handling should be meaningful. Beyond stating that something went wrong, messages resulting from exception handling should indicate why and where it went wrong. An uninformative message along the lines of "An error has occurred" only frustrates the user.

8.1.1 Try Catch and Finally Keywords

Try...Catch...Finally control structures test a piece of code and direct how the application should handle various categories of error. Each of the structure's three constituent parts plays a specific role in this process.

- The Try statement provides the code that is being tested for exceptions.
- Catch clauses identify blocks of code that are associated with specific exceptions. A Catch When block directs the code to execute under specific circumstances. A Catch without a When clause reacts to any exception. Therefore, your code might hold a series of specific Catch...When statements, each reacting to a specific type of exception, followed by a general Catch block that reacts to any exceptions that have not been intercepted by the preceding Catch...When clauses.
- The Finally statement contains code that executes regardless of whether or not an exception occurs within the Try block. A Finally statement will execute even after an Exit Try or Exit Sub. This code often performs clean-up tasks, such as closing files or clearing buffers.

8.1.2 Giving the User Information about an Exception

In exception handling, errors are dealt with in the Catch section. To do this, on the right side of Catch, declare a variable of the type of exception you want to deal with. By default, an exception is first of type Exception. Based on this, a typical formula to implement exception handling is:

```
Try
Catch ex As Exception
End Try
```

When an exception occurs in the Try section, code compilation is transferred to the Catch section. If you declare the exception as an Exception type, this class will identify the error. One of the properties of the Exception class is called Message. This property contains a string that describes the type of error that occurred. You can then use this message to display an error message if you want.

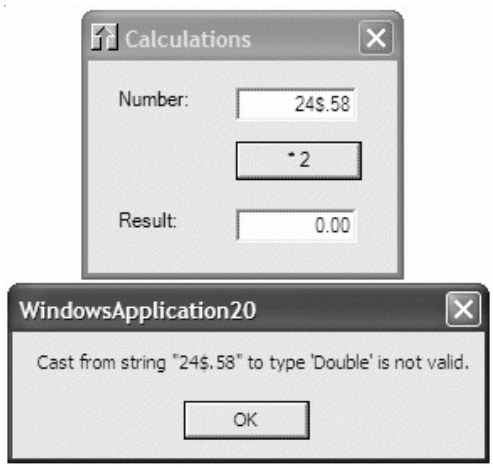
Example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
          Dim Number As Double
          Dim Twice As Double

         Try
               Number = Me.TextBox1.Text

              Twice = Number * 2

               Me.TextBox2.Text = Twice
          Catch ex As Exception
```

Notes

```
            MsgBox(ex.Message)
        End Try

    End Sub
```

Figure 8.1: Output of the Code Above



Source: http://www.functionx.com/vbnet/topics/exceptions.htm

8.1.3 Catching Only Certain Exceptions

Exception filters allow you to specify a conditional clause for each catch block. Instead of a catch block handling all exceptions of a specific type, they allow us to write catch blocks that handle exceptions of a specific type (or all exceptions) only when a certain condition is true. For example, consider the following Visual Basic .NET code:

```
Try
          ' do something that could throw an exception
Catch e As FileNotFoundException When MyFilterMethod(e)
          Console.WriteLine("first")
Catch e As FileNotFoundException When MyFilterMethod2(e)
          Console.WriteLine("second")
Catch e As FileNotFoundException When MyFilterMethod3(e)
          Console.WriteLine("third")
Catch e As DivideByZeroException
          Console.WriteLine("fourth")
End Try
```

"first" is only going to be written to the console if the try block throws a FileNotFoundException and if the MyFilterMethod returns true when passed the exception. "second" is only going to be written to the console if the try block throws a FileNotFoundException and if the MyFilterMethod2 returns true when pass the exception. etc.

8.1.4 Try-Catch Statement

Exception class represents errors that occur during application execution. This class is the base class for all exceptions. When an error occurs, either the system or the currently executing application reports it by throwing an exception containing information about the error. Once thrown, an exception is handled by the application or by the default exception handler. The common language runtime provides an exception handling model that is based on the

Notes

representation of exceptions as objects, and the separation of program code and exception handling code into try blocks and catch blocks, respectively. There can be one or more catch blocks, each designed to handle a particular type of exception, or one block designed to catch a more specific exception than another block. If an application handles exceptions that occur during the execution of a block of application code, the code must be placed within a try statement. Application code within a try statement is a try block. Application code that handles exceptions thrown by a try block is placed within a catch statement, and is called a catch block. Zero or more catch blocks are associated with a try block, and each catch block includes a type filter that determines the types of exceptions it handles. When an exception occurs in a try block, the system searches the associated catch blocks in the order they appear in application code, until it locates a catch block that handles the exception. A catch block handles an exception of type T if the type filter of the catch block specifies T or any type that T derives from. The system stops searching after it finds the first catch block that handles the exception. For this reason, in application code, a catch block that handles a type must be specified before a catch block that handles its base types, as demonstrated in the example that follows this section. A catch block that handles System.Exception is specified last. If none of the catch blocks associated with the current try block handle the exception, and the current try block is nested within other try blocks in the current call, the catch blocks associated with the next enclosing try block are searched. If no catch block for the exception is found, the system searches previous nesting levels in the current call. If no catch block for the exception is found in the current call, the exception is passed up the call stack, and the previous stack frame is searched for a catch block that handles the exception. The search of the call stack continues until the exception is handled or until no more frames exist on the call stack. If the top of the call stack is reached without finding a catch block that handles the exception, the default exception handler handles it and the application terminates.

Example:

```
Imports System

Class ExceptionTestClass

    Public Shared Sub Main()
       Dim x As Integer = 0
      Try
          Dim y As Integer = 100 / x
        Catch e As ArithmeticException
            Console.WriteLine("ArithmeticException Handler: {0}", e.ToString())
       Catch e As Exception
            Console.WriteLine("Generic Exception Handler: {0}", e.ToString())
      End Try
    End Sub 'Main
End Class 'ExceptionTestClass

Output:
ArithmeticException Handler: System.OverflowException: Arithmetic operation
resulted in an overflow. at ExceptionTestClass.Main()
```

8.1.5 Catch Blocks Using Try

No matter what kind of Exception is thrown, they all inherit from the System.Exception class. When an Exception is thrown by the .NET Framework, it automatically looks for the first Catch block (contained in a Try... Catch... block) in the stack. If the stack does not contain any Catches, an Unhandled Exception occurred. This will give your end-user a prompt saying that an Unhandled Exception occurred and that they can continue and ignore the error or quit the application.

Notes

What happens next is a mystery. In the best case, your application is closed either way. As a developer, you generally do not want users to see such a message. Therefore, it is important to always implement error handling in your code by using Try... Catch... blocks. In some cases, you might want to handle certain Exceptions differently than others. The following example provides an explanation of multiple Catches in a single Try... Catch... block. For the example, let's just say we are trying to open a file on the users' computer. Anything could happen. Maybe the directory of the file you are trying to open does not exist, maybe the file does not exist, maybe the user has no rights to open the file or maybe something completely unexpected occurs.

Example:

```
Private Sub btnMultipleCatches_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles btnMultipleCatches.Click
        Try
              ' Some code here...
               ' Let's say we're trying to open a file here.
              ' Oops! An error occurs!
              Dim rand As New Random
              If rand.Next(2) = 1 Then
                    Throw New IO.FileNotFoundException_
      ("The file you are trying to open was not found.")
            Else
                     Throw New System.Exception("An unknown error occurred.")
            End If
             ' Some more code here...
          Catch dirEx As IO.DirectoryNotFoundException
               ' Handle the specific DirectoryNotFoundException here.
               MessageBox.Show(dirEx.Message, "Directory not found", _
      MessageBoxButtons.OK, MessageBoxIcon.Error)
          Catch fileEx As IO.FileNotFoundException
               ' Handle the specific FileNotFoundException here.
               MessageBox.Show(fileEx.Message, "File not found", _
      MessageBoxButtons.OK, MessageBoxIcon.Error)
         Catch ioEx As IO.IOException
              ' Handle other non-specific IO Exceptions here.
               MessageBox.Show(ioEx.Message, "IO exception", _
      MessageBoxButtons.OK, MessageBoxIcon.Error)
         Catch ex As Exception
              ' Handle any other non-IO Exception here.
               MessageBox.Show(ex.Message, "Unknown exception", _
      MessageBoxButtons.OK, MessageBoxIcon.Error)
        End Try
    End Sub
```

Note How an Exception is generated using the Random class. 50% of the time, we will get a FileNotFoundException and 50% of the time, we will get a non-specific Exception. At this point, you might want to uncheck the checkbox you checked at the beginning of this article by using Control + Alt + E and press the 'Multiple catches' button a couple of times to see how it can show two different MessageBoxes (either the "File not found" or the "Unknown exception" MessageBoxes).

Notes

If the Exception is thrown, it will look at the first Catch block that it can use. In this case, for the FileNotFoundException this is the Catch fileEx As IO.FileNotFoundException. Notice that the FileNotFoundException skips the Catch dirEx As IO.DirectoryNotFoundException block, because it is no DirectoryNotFoundException. Also, if the fileEx has been handled in the fileEx block, it will not step into the ioEx or the ex block anymore.

Both the FileNotFoundException and the DirectoryNotFoundException inherit from the IOException. This means that if I had put the Catch ioEx As IO.IOException block above the other two Catch blocks, the FileNotFoundException and the DirectoryNotFoundException would never jump in their respective Catch blocks!

When handling Exceptions of multiple types, you must first specify the most specific Exception block. The general Catch ex as Exception should always be on the bottom of the hierarchy because every Exception can jump into this block, preventing them from jumping into more specific Catch blocks after that.

Self Assessment

Fill in the blanks:

1. An ..... is a problem that arises during the execution of a program.
2. A ..... block identifies a block of code for which particular exceptions will be activated.
3. A program catches an exception with an exception handler at the ..... block.
4. The ..... block is used to execute a given set of statements, whether an exception is thrown or not thrown.
5. A program throws an exception when a problem shows up using the ..... keyword.
6. ..... exception handling follows the exception-handling conventions of earlier versions of Visual Basic.
7. ..... exception handling handles exceptions in ways that resemble exception handling in Microsoft Visual C# and Microsoft Visual C++.
8. ..... is an event that happens during the execution of code, interrupts or disrupts the code's normal flow and creates an exception object.
9. Exception ..... allow you to specify a conditional clause for each catch block.
10. ..... class represents errors that occur during application execution.
11. A catch block that handles ..... is specified last.

8.2 The Finally Section

When an exception occurs, execution stops and control is given to the closest exception handler. This often means that lines of code you expect to always be called are not executed. Some resource cleanup, such as closing a file, must always be executed even if an exception is thrown. To accomplish this, you can use a finally block. A finally block is always executed, regardless of whether an exception is thrown. The following code example uses a try/catch block to catch an ArgumentOutOfRangeException. The Main method creates two arrays and attempts to copy one to the other. The action generates an ArgumentOutOfRangeException and the error is written to the console. The finally block executes regardless of the outcome of the copy action.

Notes

Example:

```
Imports System
Class ArgumentOutOfRangeExample

    Public Shared Sub Main()
       Dim array1 As Integer() =  {0, 0}
       Dim array2 As Integer() =  {0, 0}
      Try
           Array.Copy(array1, array2, - 1)
        Catch e As ArgumentOutOfRangeException
            Console.WriteLine("Error: {0}", e)
      Finally
            Console.WriteLine("This statement is always executed.")
      End Try
   End Sub 'Main
End Class 'ArgumentOutOfRangeExample
```

Case Study

Handling Errors in eScript

Requirements

- On **Service Agreement** list applet we have a button 'move to batch'
- User can select all contracts, or 1 or more contracts using CTRL button, and move these to his batch for executing renewals.
- User should not be able to add same contract twice in renewal batch for this 'User key' has been used and if user tries to do this he will get "*Record with same values already exists*" error
- In eScript code I want to ignore or suppress this particular error and if this comes it should execute the code for next error.

This is **absolute wrong way to do** this requirement even if you can do it. What you should be doing is to do write your script in such a way that this error doesn't come at all and there are several easy ways to ensure this.

Here is the approach that should be used to accomplish the requirement

Solution

In the **Applet server Script** write the code to implement following logic.

```
var isRecord = BC.FirstSelected();
while(isRecord)
{
    Call a workflow and pass the field values that are part of user
key
      isRecord = BC.NextSelected();
}
```

Contd...

Notes

In the Workflow we would query for the record with the values that are passed. If record is found then go to **End step** otherwise call the **Business Service** to create a new record or may be use **Insert step** and avoid all scripting all together.

The decision to use **Insert Step or BS** will depend on how complex the logic is to create the record.

If it is simple insertion then pass the values of the required field in the workflow and use it in insert step but if certain kind of data manipulation or validations are required then you can use a BS to do your job.

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://siebelunleashed.com/handling-errors-in-escript-case-study-2/?tip=1

Self Assessment

True or False:

12. When an exception occurs, execution stops and control is given to the closest exception handler.
13. A finally block is executed when called explicitly.
14. The lines of code of you expect to always be called are not executed if finally is called in between the program execution.
15. Some resource cleanup, such as closing a file, must never be executed till explicitly required.

8.3 Summary

- An exception is a problem that arises during the execution of a program.
- A Try block identifies a block of code for which particular exceptions will be activated.
- A program catches an exception with an exception handler at the place in a program where you want to handle the problem.
- The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown.
- A program throws an exception when a problem shows up.
- Unstructured Exception handling, follows the exception-handling conventions of earlier versions of Visual Basic.
- Structured Exception handling, handles exceptions in ways that resemble exception handling in Microsoft Visual C# and Microsoft Visual C++.
- Error is an event that happens during the execution of code, interrupts or disrupts the code's normal flow and creates an exception object.
- Exception filters allow you to specify a conditional clause for each catch block.
- Exception class represents errors that occur during application execution.
- A catch block that handles System.Exception is specified last.

Notes

8.4 Keywords

Catch block: A program catches an exception with an exception handler at the place in a program where you want to handle the problem.

Error: It is an event that happens during the execution of code, interrupts or disrupts the code's normal flow and creates an exception object.

Exception: It is a problem that arises during the execution of a program.

Finally block: It is used to execute a given set of statements, whether an exception is thrown or not thrown.

Structured Exception handling: It handles exceptions in ways that resemble exception handling in Microsoft Visual C# and Microsoft Visual C++.

Throw: A program throws an exception when a problem shows up.

Try block: It identifies a block of code for which particular exceptions will be activated.

Unstructured Exception handling: It follows the exception-handling conventions of earlier versions of Visual Basic.

8.5 Review Questions

1. What is exception handling?
2. Why do we need to handle exceptions?
3. What is the difference between error and exception?
4. Write a short note on try block.
5. Why do we need the catch block?
6. Explain the process of giving the user information about an exception.
7. What are exception filters?
8. Which is the parent class of all exceptions?
9. Give an example to show the use of try..catch..finally blocks.
10. Explain finally block.

Answers: Self Assessment

1.	Exception	2.	Try
3.	Catch	4.	Finally
5.	Throw	6.	Unstructured
7.	Structured	8.	Error
9.	Filter	10.	Exception
11.	System.Exception	12.	True
13.	False	14.	True
15.	False		

8.6 Further Readings

Notes

Books

Beginning Vb.Net 2003, Willis.

Object-oriented Programming with Visual Basic.Net, Hamilton.

Programming Visual Basic .NET, J. Liberty.

Visual Basic.NET Black Book, Steven Holzner.

Online links

http://www.vijaymukhi.com/documents/books/vbnet/chap7.htm

http://www.homeandlearn.co.uk/net/nets5p4.html

http://visualbasic.about.com/od/usingvbnet/a/ExHndl_2.htm

http://www.codeproject.com/Articles/9538/Exception-Handling-Best-Practices-in-NET

Notes

Unit 9: Using System.Collections

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain system.collections namespace
- Understand the arraylist class and its uses
- Discuss the advantages and disadvantages of a sorted list
- Explain LIFO and FIFO

Introduction

Collection classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. Most collection classes implement the same interfaces. Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index, etc.

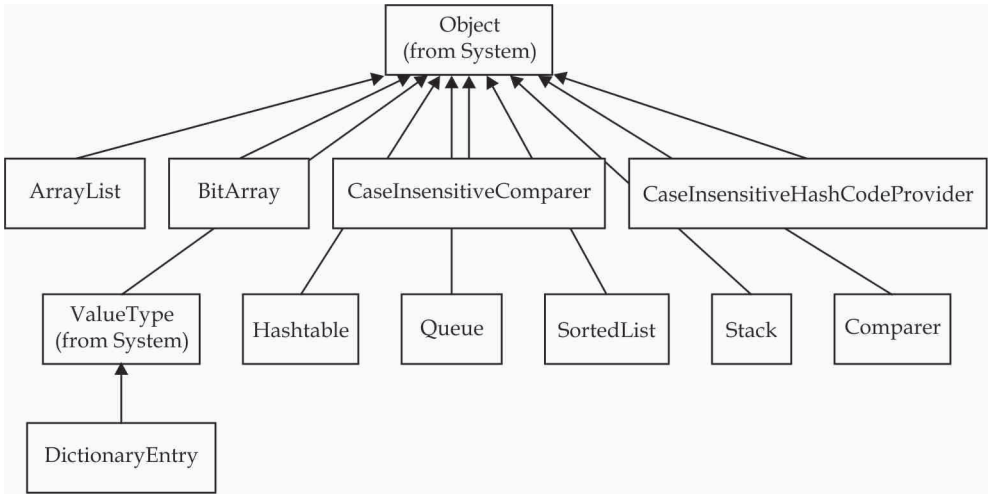
Notes

The System Namespace is contained in the file System.DLL, this Namespace is the root of the .NET Framework. It contains all the primitives: Byte, Short, Integer, Boolean, Date types and Arrays. It contains classes that can be used for data type conversion, mathematics, application environment management mathematics, and supervision of applications. The System Namespace also contains an object called the Array. An array can be like this:

```
Dim Array_Example() as Integer = {22, 46, 67}
```

In this case, the size of the Array is 3 because it contains three values. If you want to get access to the values, you need to start from 0. That being said, Array_Example(0) would have a value of 22 whereas Array_Example(2) would have a value of 67. The System.Collections Namespace has several Classes that allow you to manipulate Arrays. Classes like Comparer, which compares two objects for equivalence, and Hashtable which represents a collection of key/value pairs. The System.Collections namespace contains interfaces and classes that define various collections of objects, such as lists, queues, bit arrays, hash tables and dictionaries.

Figure 9.1: System Namespace Hierarchy



Source: http://foxcentral.net/microsoft/netforvfpdevelopers_chapter06.htm

9.1 ArrayList

The ArrayList class is an object-oriented array that can be dynamically resized at runtime. This is particularly useful where an array's dimensions cannot be changed. The ArrayList contains a variety of methods you use to manipulate array elements. ArrayList is a collection from a standard System.Collections namespace. It is a dynamic array. It provides random access to its elements. An ArrayList automatically expands as data is added. Unlike arrays, an ArrayList can hold data of multiple data types. Elements in the ArrayList are accessed via an integer index. Indexes are zero based. Indexing of elements and insertion and deletion at the end of the ArrayList takes constant time. Inserting or deleting an element in the middle of the dynamic array is more costly. It takes linear time.

9.1.1 Use of ArrayList

To add elements to an ArrayList, you use the Add method, specifying the item to be added. You can also use the Insert method to insert an element at a specified index.

```
Dim ArryList As New ArrayList()
ArryList.Add("Moe")
```

Notes

```
ArryList.Add("Larry")
ArryList.Add("Curly")
ArryList.Insert(2, "Bob")
```

This code produces elements in the following order:

```
Moe
Larry
Curly
Bob
```

You use the Remove method or the RemoveAt (specifying the index of the element to be removed) methods to remove elements from an ArrayList.

```
ArryList.Remove("Bob")
ArryList.RemoveAt(2)
```

This code produces elements in the following order:

```
Moe
Larry
```

You can access an element (get its value or set its value) at a specified index.

```
ArryList(1) = "Shemp"
```

This code sets the ArrayList elements to the following:

```
Moe
Shemp
```

9.1.2 Properties and Methods Used in ArrayList

We will summarize the properties used in ArrayList via the table below.

Table 9.1: Properties of ArrayList Class

Name	Description
Capacity	Gets or sets the number of elements that the ArrayList can contain.
Count	Gets the number of elements actually contained in the ArrayList.
IsFixedSize	Gets a value indicating whether the ArrayList has a fixed size.
IsReadOnly	Gets a value indicating whether the ArrayList is read-only.
IsSynchronized	Gets a value indicating whether access to the ArrayList is synchronized (thread safe).
Item	Gets or sets the elements at the specified index.
SyncRoot	Gets an object that can be used to synchronize access to the ArrayList.

Source: http://msdn.microsoft.com/en-us/library/vstudio/system.collections.arraylist%28v=vs.100%29.aspx

Some common methods are:

1. ***Public Overridable Function Add (value As Object) As Integer:*** Adds an object to the end of the ArrayList.
2. ***Public Overridable Sub AddRange (c As ICollection):*** Adds the elements of an ICollection to the end of the ArrayList.
3. ***Public Overridable Sub Clear:*** Removes all elements from the ArrayList.
4. ***Public Overridable Function Contains (item As Object) As Boolean:*** Determines whether an element is in the ArrayList.

Notes

5. ***Public Overridable Function GetRange (index As Integer, count As Integer) As ArrayList:*** Returns an ArrayList which represents a subset of the elements in the source ArrayList.
6. ***Public Overridable Function IndexOf (value As Object) As Integer:*** Returns the zero-based index of the first occurrence of a value in the ArrayList or in a portion of it.
7. ***Public Overridable Sub Insert (index As Integer, value As Object):*** Inserts an element into the ArrayList at the specified index.
8. ***Public Overridable Sub InsertRange (index As Integer, c As ICollection):*** Inserts the elements of a collection into the ArrayList at the specified index.
9. ***Public Overridable Sub Remove (obj As Object):*** Removes the first occurrence of a specific object from the ArrayList.
10. ***Public Overridable Sub RemoveAt (index As Integer):*** Removes the element at the specified index of the ArrayList.
11. ***Public Overridable Sub RemoveRange (index As Integer, count As Integer):*** Removes a range of elements from the ArrayList.
12. ***Public Overridable Sub Reverse:*** Reverses the order of the elements in the ArrayList.
13. ***Public Overridable Sub SetRange (index As Integer, c As ICollection):*** Copies the elements of a collection over a range of elements in the ArrayList.
14. ***Public Overridable Sub Sort:*** Sorts the elements in the ArrayList.
15. ***Public Overridable Sub TrimToSize:*** Sets the capacity to the actual number of elements in the ArrayList.

Self Assessment

True or False:

1. ArrayList is a dynamic array.
2. An ArrayList size doesn't alter when we add content to it.
3. ArrayList cannot hold data of multiple data.

9.2 HashTable

The Hashtable class represents a collection of key-and-value pairs that are organized based on the hash code of the key. It uses the key to access the elements in the collection. A hash table is used when you need to access elements by using key, and you can identify a useful key value. Each item in the hash table has a key/value pair. The key is used to access the items in the collection.

Table 9.2: Properties of HashTable Class

Property	Description
Count	Gets the number of key-and-value pairs contained in the Hashtable.
IsFixedSize	Gets a value indicating whether the Hashtable has a fixed size.
IsReadOnly	Gets a value indicating whether the Hashtable is read-only.
Item	Gets or sets the value associated with the specified key.
Key	Gets an Icollection containing the keys in the Hashtable.
Values	Gets an Icollection containing the values in the Hashtable.

Source: http://www.tutorialspoint.com/vb.net/vb.net_hashtable.htm

Notes

Table 9.3: Methods of HashTable Class

S.No.	Method Name and Purpose
1.	**Public Overridable Sub Add (key As Object, value As Object)** Adds an elements with the specified key and value into the Hashtable.
2.	**Public Overridable Sub Clear** Removes all elements from the Hashtable.
3.	**Public Overridable Function ContainsKey (key As Object) As Boolean** Determines whether the Hashtable contains a specific key.
4.	**Public Overridable Function ContainsValue (value As Object) As Boolean** Determines whether the Hashtable contains a specific value.
5.	**Public Override Sub Remove (key As Object)** Removes the element with the specified key from the Hashtable.

Source: http://www.tutorialspoint.com/vb.net/vb.net_hashtable.htm

Example:

```
Sub Main()
    Dim ht As Hashtable = New Hashtable()
   Dim k As String
    ht.Add("001", "Zara Ali")
    ht.Add("002", "Abida Rehman")
    ht.Add("003", "Joe Holzner")
    ht.Add("004", "Mausam Benazir Nur")
    ht.Add("005", "M. Amlan")
    ht.Add("006", "M. Arif")
    ht.Add("007", "Ritesh Saikia")
    If (ht.ContainsValue("Nuha Ali")) Then
          Console.WriteLine("This student name is already in the list")
   Else
         ht.Add("008", "Nuha Ali")
   End If
    ' Get a collection of the keys.
    Dim key As ICollection = ht.Keys
   For Each k In key
         Console.WriteLine(" {0} : {1}", k, ht(k))
   Next k
    Console.ReadKey()
End Sub
```

When the above code is compiled and executed, it produces following result:

```
006: M. Arif
007: Ritesh Saikia
008: Nuha Ali
003: Joe Holzner
002: Abida Rehman
004: Mausam Banazir Nur
001: Zara Ali
005: M. Amlan
```

Self Assessment

Notes

Fill in the blanks:

4. The ..... class represents a collection of key-and-value pairs that are organized based on the hash code of the key.
5. To remove elements from a HashTable use the ..... method.

9.3 Stack

It represents a last-in, first-out collection of object. It is used when you need a last-in, first-out access of items. When you add an item in the list, it is called pushing the item and when you remove it, it is called popping the item.

9.3.1 Common Functions

The common functions used in stack are summarized in Table 9.4.

Table 9.4: Methods of Stack Class

S.No.	Method Name and Purpose
1.	**Public Override Sub Clear** Removes all elements from the Stack.
2.	**Public Overridable Function Contains (obj As Object) As Boolean** Determines whether as elements in the Stack.
3.	**Public Overridable Function Peek As Object** Returns the object at the top of the Stack without removing it.
4.	**Public Overridable Function Pop As Object** Removes and returns the object at the top of the Stack.
5.	**Public Overridable Sub Push (obj As Object)** Inserts an object at the top of the Stack.
6.	**Public Overridable Function ToArray As Object()** Copies the Stack to a new array.

Source: http://www.tutorialspoint.com/vb.net/vb.net_stack.htm

9.3.2 Implementation

The following example demonstrates use of Stack:

```
Sub Main()
      Dim st As Stack = New Stack()
      st.Push("A")
      st.Push("M")
      st.Push("G")
      st.Push("W")
       Console.WriteLine("Current stack: ")
     Dim c As Char
     For Each c In st
          Console.Write(c + " ")
    Next c
      Console.WriteLine()
```

Notes

```
        st.Push("V")
        st.Push("H")
         Console.WriteLine("The next poppable value in stack: {0}", st.Peek())
          Console.WriteLine("Current stack: ")
       For Each c In st
            Console.Write(c + " ")
       Next c
         Console.WriteLine()
          Console.WriteLine("Removing values ")
       st.Pop()
       st.Pop()
       st.Pop()
         Console.WriteLine("Current stack: ")
       For Each c In st
            Console.Write(c + " ")
       Next c
        Console.ReadKey()
    End Sub
```

When the above code is compiled and executed, it produces following result:

```
Current stack:
W G M A
The next poppable value in stack: H
Current stack:
H V W G M A
Removing values
Current stack:
G M A
```

9.3.3 Usage Scenarios

There are many uses of a stack. Some of them can be used to evaluate expressions like Postfix to Prefix etc.

Self Assessment

True or False:

6. Stack follows FIFO mechanism.
7. When you add an item in the stack list, it is called pushing the item.

9.4 Queue

A queue is a First-In-First-Out (FIFO) data structure. The first element added to the queue will be the first one to be removed. Queues may be used to process messages as they appear or serve customers as they come. The first customer which comes should be served first.

9.4.1 Common Functions

We will now summarize the commonly used methods in queues in Table 9.5.

Notes

Table 9.5: Methods of Queue Class

Name	Description
Clear	Removes all objects from the Queue.
Clone	Creates a shallow copy of the Queue.
Contains	Determines whether an element is in the Queue.
CopyTo	Copies the Queue elements to an existing one-dimensional Array, starting at the specified array index.
Dequeue	Removes and returns the object at the beginning of the Queue.
Enqueue	Adds an object to the end of the Queue.
Equals(Object)	Determines whether the specified object is equal to the current object. (Inherited from Object.)
GetEnumerator	Returns and enumerator that iterates through the Queue.
GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
GetType	Gets the Type of the current instance. (Inherited from Object.)
MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)

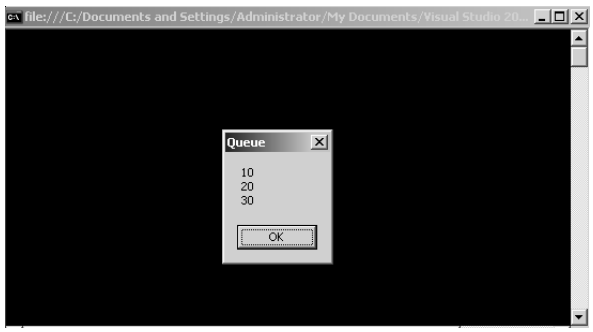
Source: http://msdn.microsoft.com/en-us/library/system.collections.queue.aspx

9.4.2 Use of Queue

We will now discuss the implementation of a queue with an example.

```
Imports System.Windows.Forms
Module Module1
     Sub Main()
          Dim nameQueue As New Queue(Of String)
          nameQueue.Enqueue("10")
          nameQueue.Enqueue("20")
          nameQueue.Enqueue("30")
          Dim nameQueueString As String = ""
          Do While nameQueue.Count > 0
               nameQueueString &= nameQueue.Dequeue & vbCrLf
         Loop
           MessageBox.Show(nameQueueString, "Queue")
          Console.ReadLine()
     End Sub
End Module
```

Figure 9.2: Output of the Code Above



Source: http://www.dotnetheaven.com/article/queues-and-stacks-using-vb.net

Notes

Self Assessment

Fill in the blanks:

8. A queue is a ..... data structure.

9. ..... is the process of adding items to a queue.

9.5 SortedList

The SortedList class represents a collection of key-and-value pairs that are sorted by the keys and are accessible by key and by index. A sorted list is a combination of an array and a hash table. It contains a list of items that can be accessed using a key or an index. If you access items using an index, it is an ArrayList, and if you access items using a key, it is a Hashtable. The collection of items is always sorted by the key value.

Example:

```
Public Shared Sub Main()

          ' Creates and initializes a new SortedList.
          Dim mySL As New SortedList()
          mySL.Add("Third", "!")
          mySL.Add("Second", "World")
          mySL.Add("First", "Hello")

          ' Displays the properties and values of the SortedList.
          Console.WriteLine("mySL")
          Console.WriteLine("   Count:     {0}", mySL.Count)
          Console.WriteLine("   Capacity: {0}", mySL.Capacity)
          Console.WriteLine("   Keys and Values:")
          PrintKeysAndValues(mySL)
     End Sub

      Public Shared Sub PrintKeysAndValues(myList As SortedList)
            Console.WriteLine(ControlChars.Tab & "-KEY-" & ControlChars.Tab &
_
            "-VALUE-")
         Dim i As Integer
         For i = 0 To myList.Count - 1
               Console.WriteLine(ControlChars.Tab & "{0}:" & ControlChars.Tab
& _
                   "{1}", myList.GetKey(i), myList.GetByIndex(i))
         Next i
          Console.WriteLine()
     End Sub

Output:

mySL
```

Notes

```
Count:     3
Capacity:  16
Keys  and  Values:
-KEY-       -VALUE-
First:      Hello
Second:     World
Third:      !
```

9.5.1 Advantages and Disadvantages of Using a SortedList

System.Collections.SortedList type is a key-based dictionary type that allows you to store items in an ordered manner. This is advantageous whenever you need to be able to store sorted data without going through a separate routine of sorting the contents of a collection. The convenience of a SortedList comes at the price of slower additions and removals of items to the list. A SortedList utilizes two arrays internally: one array stores keys, while another stores the data or object reference.

When you work with SortedLists in VB.NET, keep in mind that objects are sorted based on the key value and not an object itself; therefore, the key needs to be a type that supports IComparable–either an integer or a string. Otherwise, you have to create a custom IComparer as an argument to the SortedList when defining the SortedList.

Self Assessment

Fill in the blanks:

10. The SortedList class represents a collection of key-and-value pairs that are sorted by the ..... and are accessible by key and by index.

11. A SortedList utilizes two arrays internally: one array stores keys, while another stores the ..... or ..... reference.

9.6 BitArray

The BitArray class manages a compact array of bit values, which are represented as Booleans, where true indicates that the bit is on (1) and false indicates the bit is off (0). It is used when you need to store the bits but do not know the number of bits in advance. You can access items from the BitArray collection by using an integer index, which starts from zero.

Some of the common properties are shown in Table 9.6:

Table 9.6: Properties of BitArray Class

Property	Description
Count	Gets the number of elements contained in the BitArray.
IsReadOnly	Gets a value indicating whether the BitArray is read-only.
Item	Gets or sets the value of the bit at a specific position in the BitArray.
Length	Gets or sets the number of elements in the BitArray.

Source: http://www.tutorialspoint.com/vb.net/vb.net_bitarray.htm

Notes

The list of common methods used in the BitArray class is given below.

Table 9.7: Methods of BitArray Class

S.No.	Method Name and Purpose
1.	**Public Function And (value As BitArray) As BitArray** Performs the bitwise AND operation on the elements in the current BitArray against the corresponding elements in the Specified BitArray.
2.	**Public Function Get (Index As Integer) As Boolean** Gets the value on the bit at a specific position in the BitArray.
3.	**Public Function Not As BitArray** Inverts all the bit values in the current BitArray, so that elements set to true are changed to false, and elements set to false are changed to true.
4.	**Public Function Or (value As BitArray) As BitArray** Performs the bitwise OR operation on the elements in the current BitArray against the corresponding elements in the specified BitArray.
5.	**Public Sub Set (Index As Integer, value As Boolean)** Sets the bit at a specific position in the BitArray to the specified value.
6.	**Public Sub SetAll (value As Boolean)** Sets all bits in the BitArray to the specified value.

Source: http://www.tutorialspoint.com/vb.net/vb.net_bitarray.htm

Example:

```
Sub Main()
        'creating two  bit arrays of size 8
        Dim ba1 As BitArray = New BitArray(8)
        Dim ba2 As BitArray = New BitArray(8)
       Dim a() As Byte = {60}
       Dim b() As Byte = {13}
        'storing the values 60, and 13 into the bit arrays
       ba1 = New BitArray(a)
       ba2 = New BitArray(b)
       'content of ba1
        Console.WriteLine("Bit array ba1: 60")
       Dim i As Integer
       For i = 0 To ba1.Count
             Console.Write("{0 } ", ba1(i))
      Next i
        Console.WriteLine()
       'content of ba2
        Console.WriteLine("Bit array ba2: 13")
       For i = 0 To ba2.Count
             Console.Write("{0 } ", ba2(i))
      Next i
        Console.WriteLine()
        Dim ba3 As BitArray = New BitArray(8)
       ba3 = ba1.And(ba2)
       'content of ba3
        Console.WriteLine("Bit array ba3 after AND operation: 12")
       For i = 0 To ba3.Count
             Console.Write("{0 } ", ba3(i))
```

Notes

```
      Next i
       Console.WriteLine()
      ba3 = ba1.Or(ba2)
      'content of ba3
       Console.WriteLine("Bit array ba3 after OR operation: 61")
      For i = 0 To ba3.Count
           Console.Write("{0 } ", ba3(i))
      Next i
       Console.WriteLine()
       Console.ReadKey()
   End Sub
```

Self Assessment

True or False:

12. The BitArray class manages a compact array of bit values, which are represented as Booleans.
13. You can access items from the BitArray collection by using an integer index, which starts from one.

9.7 BitVector32

The BitVector32 structure is in the System.Collections.Specialized namespace. Essentially, it is a single array (vector) of 32 bits, where a value of one corresponds to true (on) and a value of zero is false (off). By default, when creating a new BitVector32 object, you will start with all of the bits turned off. As an alternative, you can pass a parameter into the constructor to turn on some of the bits. If you want all of the bits turned on, pass the value of -1, which is the bitwise complement to zero.

Example:

```
Imports System
Imports System.Collections.Specialized

Public Class SamplesBitVector32

    Public Shared Sub Main()

        ' Creates and initializes a BitVector32 with all bit flags set to
FALSE.
       Dim myBV As New BitVector32(0)

        ' Creates masks to isolate each of the first five bit flags.
        Dim myBit1 As Integer = BitVector32.CreateMask()
        Dim myBit2 As Integer = BitVector32.CreateMask(myBit1)
        Dim myBit3 As Integer = BitVector32.CreateMask(myBit2)
        Dim myBit4 As Integer = BitVector32.CreateMask(myBit3)
        Dim myBit5 As Integer = BitVector32.CreateMask(myBit4)

        ' Sets the alternating bits to TRUE.
        Console.WriteLine("Setting alternating bits to TRUE:")
       Console.WriteLine("    Initial:            {0}", myBV.ToString())
```

Notes

```
        myBV(myBit1) = True
         Console.WriteLine("    myBit1 = TRUE:    {0}", myBV.ToString())
        myBV(myBit3) = True
         Console.WriteLine("    myBit3 = TRUE:    {0}", myBV.ToString())
        myBV(myBit5) = True
         Console.WriteLine("    myBit5 = TRUE:    {0}", myBV.ToString())
    End Sub 'Main
End Class 'SamplesBitVector32


Output:

Setting alternating bits to TRUE:
Initial:            BitVector32{00000000000000000000000000000000}
myBit1 = TRUE:    BitVector32{00000000000000000000000000000001}
myBit3 = TRUE:    BitVector32{00000000000000000000000000000101}
myBit5 = TRUE:    BitVector32{00000000000000000000000000010101}
```

Case Study

Stack Overflow

Situation

Two prominent bloggers in the world of software development wanted to create a Question and Answer Web site that would provide a central repository for accumulating programming wisdom as developers posted questions and their peers posted replies.

Jeff Atwood, developer and author of the popular programming and human factors blog Coding Horror; and Joel Spolsky, developer and the author of the software development blog Joel on Software, and founder of Fog Creek Software, envisioned their Web site as a community-run organization similar to Wikipedia. The two also saw the site as incorporating aspects of blogging as well as community voting in the spirit of the Digg and Reddit social news Web sites in which the prominence of information is based upon user ranking of its relevance.

The mission of the site was summed up in a blog entry by Atwood that reads: "None of Us is as Dumb as All of Us."

Atwood and Spolsky wanted to create a site that behind the scenes was simple and fast, with code that was lean and scalable. They were dissatisfied with traditional Web development approaches that used an abstraction layer and that embedded code on Web pages to preserve view states. In addition to potentially slowing response rates, they were concerned that storing code on pages could adversely impact search engine optimization because view state information at the top of a page could potentially lessen the relevance a search engine accords to the actual page content found lower on the page.

The two developers wanted to create their new solution using a Model-View-Controller (MVC) approach, which simplifies application programming. MVC architecture, first described by Trygve Reekskaug in 1979, uses a design pattern based upon a model which provides a data representation relevant to the needs of the application; a view, or interface, for interacting with the data model; and a controller for mediating between the view and the model objects. The three together are sometimes referred to as a triplet,

Contd...

Notes

and MVC-based applications can be assembled as a series of triplets, each independent of the other.

The two wanted to create their site using the Microsoft Application Platform, but needed an MVC solution. "We were definitely sticking with the Microsoft stack because we like the Microsoft stack," Atwood says. "We are big fans of the language C#. Historically C# has evolved rapidly, and in ways that we like. To stay within C# we were preparing to create our own MVC implementation, but then we heard about the beta version of the Microsoft ASP.NET MVC Framework."

Solution

Atwood and Spolsky created their new site–Stack Overflow–using the ASP.NET MVC Framework, the Visual Studio 2008 development system, and the Microsoft .NET Framework 3.5, which are all part of the Microsoft Web Platform. The site, which was created within two months, proved popular and was able to scale to support what turned out to be rapid growth.

"Stack Overflow has grown incredibly fast," Spolsky says. "After a year in business, it gets over a million page views most weekdays and currently stands as the 817th largest site on the Internet, according to Quantcast. It reaches 5.2 million people a month." At last count the site hosted 500,000 questions along with the related answers and discussions, ranked according to what the community felt were the best responses.

The two have repurposed the same code to broaden the scope beyond programmers, launching sister sites including Server Fault, for system administrators, and Super User, for computer "power users."

A spin off of the software, called StackExchange, has been released by Fog Creek Software to allow third parties to build Stack-Overflow communities around other topics. "You can use the same software to create a Q&A site about anything," Atwood says. "Someone could create a Stack Overflow Q&A community for airplanes, or stamp collecting, geology, or any other area of interest."

Stack Overflow was created using the Microsoft Web Platform, which in addition to development tools includes the Windows Server 2008 operating system, Internet Information Services (IIS) 7, and Microsoft SQL Server 2008 database software.

The basic elements of Stack Overflow include:

- ***Load Balancing.*** Users entering the stackoverflow.com site are first handled by a load balancer, which assigns a user to one of several Web servers according to IP address. Users continue to be assigned to the same server for as long as they have the same IP address. Stack Overflow uses the open-source HAProxy for this load balancing.
- ***Presentation.*** Customers browse pages served by Windows Server 2008 Standard, through IIS 7.5. The Web servers are computers with Xeon quad core CPUs and 8 gigabytes (GB) RAM. When a user submits a question or an answer, their information is written to the database using the lightweight LINQ to SQL database abstraction layer.
- ***Database.*** The database is hosted on a dedicated server with 2 Intel Xeon quad core CPUs and 48 GB of RAM, and runs Microsoft SQL Server 2008 Enterprise. Stack Overflow takes advantage of the Online Indexing feature, introduced with SQL Server 2008 Enterprise, which enables table re-indexing while the database continues to run.

Contd...

Notes

"We designed this to be a next generation Q&A site," Atwood says. "We tried to combine aspects of Wikis, in that everybody can edit everything once the system learns to trust them. Voting, as with DIGG and REDDIT, are an important component. Stack Overflow has a reputation system which comes out of the voting, so as other people vote your content up that increases your reputation score and it unlocks different abilities on the site as the system learns to trust you. It has elements of blogging in that there is a strong sense of ownership. When you create a question you are the owner of that question and you have special privileges as you curate that question. And, like a blog, people can post comments. We wanted to synthesize the best of what we saw in all of these types of communities and bring them all together into one simple site."

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://www.microsoft.com/india/casestudies/microsoft-visual-studio-2008/stack-overflow/developers-see-faster-web-coding-better-performance-with-model-view-controller/4000006676

Self Assessment

Fill in the blanks:

14. The BitVector32 structure is in the ..... namespace.
15. By default, when creating a new BitVector32 object, you will start with all of the bits turned ..... .

9.8 Summary

- Collection classes are specialized classes for data storage and retrieval.
- The System Namespace is contained in the file System.DLL, this Namespace is the root of the .NET Framework.
- The ArrayList class is an object-oriented array that can be dynamically resized at runtime.
- ArrayList is a dynamic array that automatically expands as data is added.
- The Hashtable class represents a collection of key-and-value pairs that are organized based on the hash code of the key.
- Stack represents a last-in, first-out collection of object.
- When you add an item in the list, it is called pushing the item and when you remove it, it is called popping the item.
- A queue is a First-In-First-Out (FIFO) data structure.
- The SortedList class represents a collection of key-and-value pairs that are sorted by the keys and are accessible by key and by index.
- The BitArray class manages a compact array of bit values, which are represented as Booleans.
- The BitVector32 structure is in the System.Collections.Specialized namespace.
- By default, when creating a new BitVector32 object, you will start with all of the bits turned off.

Notes

9.9 Keywords

ArrayList Class: It is an object-oriented array that can be dynamically resized at runtime.

BitArray Class: It manages a compact array of bit values, which are represented as Booleans.

Collection Classes: They are specialized classes for data storage and retrieval.

Hashtable Class: It represents a collection of key-and-value pairs that are organized based on the hash code of the key.

Queue: It is a First-In-First-Out (FIFO) data structure.

SortedList Class: It represents a collection of key-and-value pairs that are sorted by the keys and are accessible by key and by index.

Stack: It represents a last-in, first-out collection of object.

System.DLL: This Namespace is the root of the .NET Framework.

9.10 Review Questions

1. What is an ArrayList?
2. Explain the use of ArrayList.
3. List the common properties and methods used in ArrayList.
4. Write a note on HashTable.
5. What is the basic use of a Stack.
6. List the common functions of a stack.
7. What mechanism does a Queue use to add or delete an element?
8. Explain the concept of a SortedList.
9. List the advantages and disadvantages of using a SortedList.
10. Differentiate between an Array and a BitArray.

Answers: Self Assessment

1. True
2. False
3. False
4. HashTable
5. Clear
6. False
7. True
8. LIFO
9. Enqueue
10. Keys
11. Data , Object
12. True
13. False
14. System.Collections.Specialized
15. Off

Notes

9.11 Further Readings

Books

Beginning Vb.Net 2003, Willis.

Object-oriented Programming with Visual Basic.Net, Hamilton.

Programming Visual Basic .NET, J. Liberty.

Visual Basic.NET Black Book, Steven Holzner.

Online links

http://www.functionx.com/vb/collections/introduction.htm

http://msdn.microsoft.com/en-us/library/system.collections.aspx

http://net.blogs.webucator.com/2010/12/01/net-framework-using-the-bitvector32-structure/

http://www.tutorialspoint.com/vb.net/vb.net_queue.htm

Unit 10: Windows Programming

Notes

CONTENTS

Notes

Objectives

After studying this unit, you will be able to:

- Understand textboxes and rich textboxes
- Explain box property
- Discuss date time picker
- Explain comboboxes
- Understand button creation
- Describe rich textboxes

Introduction

Windows Forms is a framework located in the System.Windows.Forms.dll assembly for building Windows applications in .NET based on a Graphical User Interface (GUI). Any language that supports the Common Language Runtime (CLR) can use Windows Forms. If you have programmed in Visual Basic (VB), you are probably familiar with forms. In VB, all windows are forms. Controls are placed on forms to develop GUI applications. Visual C++ developers will more likely be familiar with windows and dialogs rather than forms (CWnd and CDialog in Microsoft Foundation Classes [MFC]). The Microsoft .NET Framework is designed to remedy this "forms versus windows" situation. All windows are forms, including dialog boxes. From all of this synergy, Microsoft coined the term Windows form. Now developers using any .NET-supported language have access to the same windowing classes, whether they work with C#, VB, C++, or any other .NET-compliant language. This language independence has been extended to support many more languages, including COBOL. In addition to the preceding, the main benefits of Windows Forms are its ease of use, the standardization of the control hierarchy, and that it allows for Rapid Application Development (RAD). Changing the colors and fonts of controls using MFC or Win32 can be a real headache. The .NET Framework has taken care of most such problems and inconveniences.

In addition, Windows Forms applications provide the following:

- Simple and flexible property support
- Common control support, including support for font and color dialogs
- Support for Web Services
- Data-aware controls using ADO.NET
- ActiveX support
- GDI+ (Graphical Device Interface +), a better and richer graphics library, which supports alpha blending, texture brushes, advanced transformations, and rich text
- Metadata support

10.1 Controls

The Control Class defines the base class for controls, which are components with visual representation.

Notes

```
Syntax:
<ClassInterfaceAttribute(ClassInterfaceType.AutoDispatch)> _
<ComVisibleAttribute(True)> _
Public Class Control _
   Inherits Component _
   Implements IDropTarget, ISynchronizeInvoke, IWin32Window,
IBindableComponent, _
   IComponent, IDisposable
```

To create your own control class, inherit from the UserControl, Control classes, or from the other Windows Forms provided controls. The Control class implements very basic functionality required by classes that display information to the user. It handles user input through the keyboard and pointing devices. It handles message routing and security. It defines the bounds of a control (its position and size), although it does not implement painting. It provides a window handle (hWnd). Windows Forms controls use ambient properties so child controls can appear like their surrounding environment. An ambient property is a control property that, if not set, is retrieved from the parent control. If the control does not have a Parent, and the property is not set, the control attempts to determine the value of the ambient property through the Site property. If the control is not sited, if the site does not support ambient properties, or if the property is not set on the AmbientProperties, the control uses its own default values. Typically, an ambient property represents a characteristic of a control, such as BackColour, that is communicated to a child control. For example, a Button will have the same BackColor as its parent Form by default. Ambient properties provided by the Control class include: Cursor, Font, BackColor, ForeColor, and RightToLeft.

10.1.1 Control Tab Order

Using the Tab key we can move focus from one control to other. We can also customize this tab order. For this we make use of the Tab Index property. If two controls have same value of the TabIndex then the focus first goes to the control that is closest to the front of the form. From the GUI the Tab Index from the View menu enables us to do the same action.

Self Assessment

True or False:

1. Windows Forms is a framework located in the System.Windows.Forms.dll assembly.
2. The FormControl Class defines the base class for controls.
3. Tab property is set so that we can move focus from one control to other.

10.2 Button Control

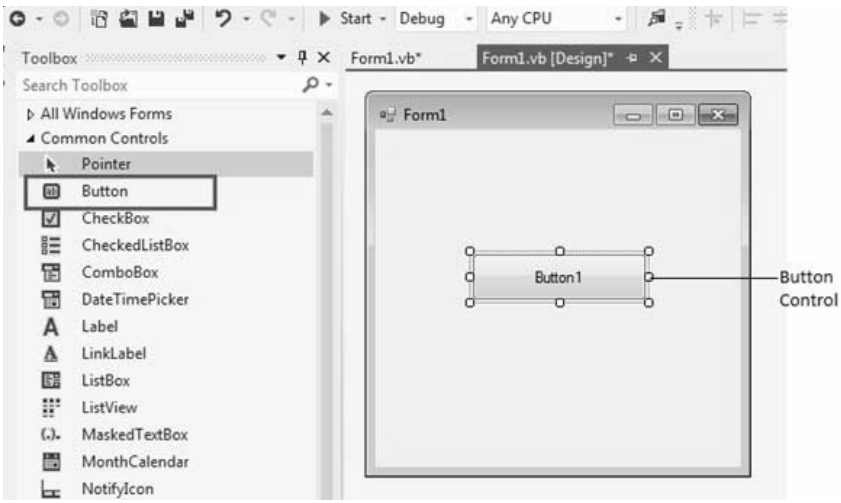
The Button control represents a standard Windows button. It is generally used to generate a Click event by providing a handler for the Click event.

Example:

We can create a label by dragging a Button control from the Toolbox and dropping it on the form.

Notes

Figure 10.1: Button Control



Source: http://www.tutorialspoint.com/vb.net/vb.net_button.htm

The following are some of the commonly used properties of the Button control:

Table 10.1: Properties of Button

S.No.	Property	Description
1.	AutoSizeMode	Gets or sets the mode by which the Button automatically resizes itself.
2.	BackColor	Gets or sets the background color of the control.
3.	BackgroundImage	Gets or sets the background image displayed in the control.
4.	DialogResult	Gets or sets a value that is returned to the parent form when the button is clicked. This is used while creating dialog boxes.
5.	ForeColor	Gets or sets the foreground color of the control.
6.	Image	Gets or sets the image that is displayed on a button control.
7.	Location	Gets or sets the coordination of the upper-left corner of the control relative to the upper-left corner of its container.
8.	TabIndex	Gets or sets the tab order of the control within container.
9.	Text	Gets or sets the text associated with this control.

Source: http://www.tutorialspoint.com/vb.net/vb.net_button.htm

The following are some of the commonly used methods of the Button control:

Table 10.2: Methods of Button

S.No.	Method Name and Description
1.	**GetPreferredSize** Retrieves the size of a rectangular area in which a control can be fitted.
2.	**NotifyDefault** Notifies the Buttom whether it is the default button so that it can adjust its appearance accordingly.
3.	**Select** Activates the control.
4.	**ToString** Returns a String containing the name of the Component, if any. This method should not be overridden.

Source: http://www.tutorialspoint.com/vb.net/vb.net_button.htm

Notes

The following are some of the commonly used events of the Button control:

Table 10.3: Events of Button

S.No.	Event	Description
1.	Click	Occurs when the control is clicked.
2.	DoubleClick	Occur when the user double-clicks the Button control.
3.	GotFocus	Occurs when the control receives focus.
4.	TabIndexChanged	Occur when the TabIndex Property value changes.
5.	TextChanged	Occurs when the Text property value changes.
6.	Validated	Occurs when the control is finished validating.

Source: http://www.tutorialspoint.com/vb.net/vb.net_button.htm

10.2.1 Button Event

Click event is the default event of a button. When a Button is clicked it responds with the Click Event.

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

// Code

End Sub
```

10.2.2 Working with Buttons

We can set properties of a button. Let us show this with an example.

```
Public Class Form1
   Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        ' Set the caption bar text of the form.
        Me.Text = "abt.com"
        btnImage.Visible = False
    End Sub
     Private Sub btnMoto_Click(sender As Object, e As EventArgs) Handles
btnMoto.Click
        btnImage.Visible = False
        Label1.Text = "Simple Easy Learning"
    End Sub
     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles
btnExit.Click
        Application.Exit()
    End Sub
     Private Sub btnLogo_Click(sender As Object, e As EventArgs) Handles
btnLogo.Click
        Label1.Visible = False
        btnImage.Visible = True
    End Sub
End Class
```

Notes

Self Assessment

Fill in the blanks:

4. ..... method of a button is used to activate a control.

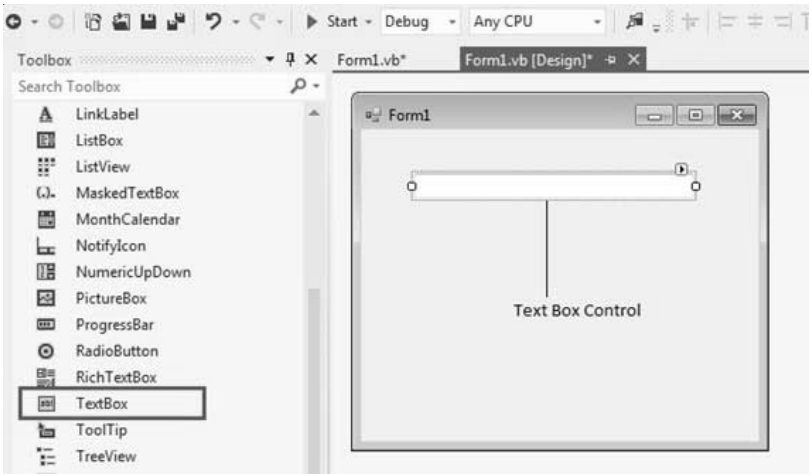
5. ..... event is the default event of a button.

10.3 TextBox Control

The **TextBox** control in Visual Basic 6.0 is replaced by the Windows Forms **TextBox** control in Visual Basic .NET. The names of some properties, methods, events, and constants are different, and in some cases there are differences in behavior. The following tables list Visual Basic 6.0 properties, methods, and events and their Visual Basic .NET equivalents. Where applicable, constants are indented beneath the property or method. All Visual Basic .NET constants map to the **System.Windows.Forms** namespace unless otherwise noted. Text box controls allow entering text on a form at runtime. By default, it takes a single line of text, however, you can make it accept multiple texts and even add scroll bars to it.

Example: Let's create a text box by dragging a TextBox control from the Toolbox and dropping it on the form.

Figure 10.2: TextBox Control



Source: http://www.tutorialspoint.com/vb.net/vb.net_textbox.htm

The following are some of the commonly used properties of the TextBox control:

Table 10.4: TextBox Properties

AutoCompleteSource	Gets or sets a value specifying the source of complete strings used for automatic completion.
CharacterCasing	Gets or sets whether the TextBox control modifies the case of characters as they are typed.
Font	Gets or sets the font of the text displayed by the control.
FontHeight	Gets or sets the height of the font of the control.
Forecolor	Gets or sets the foreground color of the control.

Contd...

Notes

Lines	Gets or sets the lines of text in a text box control.
Multiline	Gets or sets a value indicating whether this is a multiline TextBox control.
PasswordChar	Gets or sets the character used to mask characters of a password in a single-line TextBox control.
ReadOnly	Gets or sets a value indicating whether text in the text box is read-only.
ScrollBars	Gets or sets which scroll bars should appear in a multiline TextBox control. This property has values: • None • Horizontal • Vertical • Both
TabIndex	Gets or sets the tab order of the control within its container.
Text	Gets or sets the current text in the TextBox.
TextAlign	Gets or sets how text is aligned in a TextBox control. This property has values: • Left • Right • Center
TextLength	Gets the length of text in the control.
WordWrap	Indicates whether a multiline text box control automatically wraps words to the beginning of the next line when necessary.

Source: http://www.tutorialspoint.com/vb.net/vb.net_textbox.htm

The following are some of the commonly used methods of the TextBox control:

Table 10.5: TextBox Methods

S.No.	Method Name and Description
1.	**AppendText** Appends text to the current text of a textbox.
2.	**Clear** Clears all text from the textbox control.
3.	**Copy** Copies the current selection in the textbox to the Clipboard.
4.	**Cut** Moves the current selection in the textbox to the Clipboard.
5.	**Paste** Replaces the current selection in the textbox with the contents of the Clipboard.
6.	**Paste(String)** Sets the selected to the specified text without clearing the undo buffer.
7.	**ResetText** Resets the Text property to its default value.
8.	**ToString** Returns a string that represents the TextBoxBase control.
9.	**Undo** Undoes the last edit operation in the textbox.

Source: http://www.tutorialspoint.com/vb.net/vb.net_textbox.htm

Notes

The following are some of the commonly used events of the Text control:

Table 10.6: TextBox Events

S.No.	Event	Description
1.	Click	Occurs when the control is clicked.
2.	DoubleClick	Occurs when the control is double-clicked.
3.	TextAlighChanged	Occurs when the TextAlign property value changes.

Source: http://www.tutorialspoint.com/vb.net/vb.net_textbox.htm

Example:

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) _
        Handles MyBase.Load
       ' Set the caption bar text of the form.
       Me.Text = "tutorialspont.com"
   End Sub

    Private Sub btnMessage_Click(sender As Object, e As EventArgs) _
    Handles btnMessage.Click
       MessageBox.Show("Thank you " + txtName.Text + " from " + txtOrg.Text)
   End Sub
End Class
```

Self Assessment

Fill in the blanks:

6. All Visual Basic .NET constants map to the ..... namespace unless otherwise noted.
7. TextAlign can take up three values: ..... , ... or ..... .

10.4 RichTextBox

A RichTextBox control is an advanced text box that provides text editing and advanced formatting features including loading Rich Text Format (RTF) files.

We can create a RichTextBox control using a Forms designer at design-time or using the RichTextBox class in code at run-time. To create a RichTextBox control at design-time, you simply drag and drop a RichTextBox control from Toolbox onto a Form in Visual Studio. Once a RichTextBox is added to a Form, you can move it around and resize it using mouse and set its properties and events. Creating a RichTextBox control at run-time is merely a work of creating an instance of RichTextBox class, set its properties and add RichTextBox class to the Form controls.

First step to create a dynamic RichTextBox is to create an instance of RichTextBox class. The following code snippet creates a RichTextBox control object.

```
Dim dynamicRichTextBox As New RichTextBox()
```

In the next step, you may set properties of a RichTextBox control. The following code snippet sets size, location, background color, foreground color, Text, Name, and Font properties of a RichTextBox.

Notes

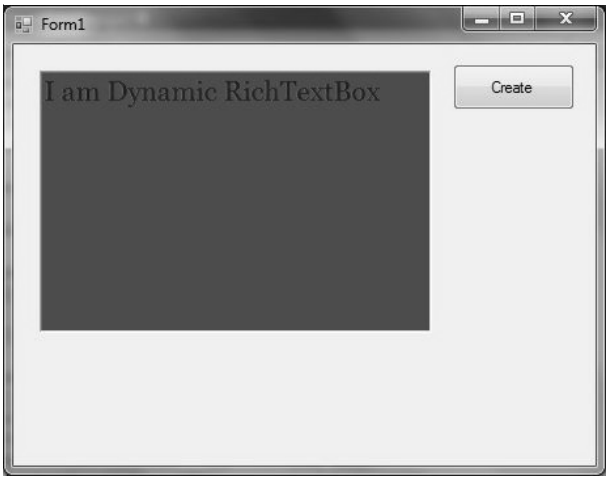
```
dynamicRichTextBox.Location = New Point(20, 20)
dynamicRichTextBox.Width = 300
dynamicRichTextBox.Height = 200
'   Set background and foreground
dynamicRichTextBox.BackColor = Color.Red
dynamicRichTextBox.ForeColor = Color.Blue
dynamicRichTextBox.Text = "I am Dynamic RichTextBox"
dynamicRichTextBox.Name = "DynamicRichTextBox"
dynamicRichTextBox.Font = New Font("Georgia", 16)
```

Once a RichTextBox control is ready with its properties, next step is to add the RichTextBox control to the Form. To do so, we use Form.Controls.Add method. The following code snippet adds a RichTextBox control to the current Form.

```
Controls.Add(dynamicRichTextBox)
```

RichTextBox control looks like Figure below.

Figure 10.3: RichTextBox Control



Source: http://www.dotnetheaven.com/article/richtextbox-control-in-vb.net

Self Assessment

True or False:

8. A RichTextBox control is an advanced text box that provides text editing and advanced formatting features including loading Rich Text Format (RTF) files.
9. Creating a RichTextBox control at run-time is done by creating an instance of Control class.

10.5 Label and LinkLabel

We will now discuss two controls that are used for data display in a VB.NET form.

10.5.1 Label

Microsoft Visual Studio .NET controls are the graphical tools you use to build the user interface of a VB.Net program. Labels are one of the most frequently used Visual Basic control. A Label control lets you place descriptive text, where the text does not need to be changed by the user. The Label class is defined in the System.Windows.Forms namespace.

Notes

Figure 10.4: Label Control

Source: http://vb.net-informations.com/gui/vb.net-label.htm

Add a Label control to the form. Click Label in the Toolbox and drag it over the forms Designer and drop it in the desired location.

If you want to change the display text of the Label, you have to set a new text to the Text property of Label.

```
Label1.Text = "This is my first Label"
```

You can load Image in Label control, if you want to load an Image in the Label control you can code like this

```
Label1.Image = Image.FromFile("C:\testimage.jpg")
```

The following source code shows how to set some properties of the Label through coding.

Example:

```
Public Class Form1
      Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
          Label1.Text = "This is my first Label"
           Label1.BorderStyle = BorderStyle.FixedSingle
           Label1.TextAlign = ContentAlignment.MiddleCenter
     End Sub
End Class
```

10.5.2 LinkLabel

The VB.NET LinkLabel control is similar to the label control, with the exception of hyperlinks. The LinkLabel control can display single or multiple hyperlinks. When the user clicks a hyperlink, the process can be started in the event.

```
Namespace: System::Windows::Forms
System.Windows.Forms (in system.windows.forms.dll)
```

Creating an new VB.NET Windows forms LinkLabel.

```
Dim linklabel As New LinkLabel
```

Assigning a AutoSize Value to the LinkLabel Control.

```
linklabel.AutoSize = False
```

Docking the VB.NET LinkLabel Control.

```
linklabel.Dock = DockStyle.Top
```

Assigning a text value to the VB.NET LinkLabel control.

```
linklabel.Text = "Link to Web Site"
```

Adding a VB.NET LinkLabel control to the form.

```
Me.Controls.Add(linklabel)
```

Notes

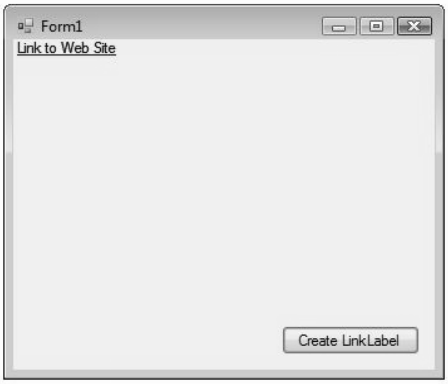
Creating a LinkClicked Event for the VB.NET LinkLabel control.

```
AddHandler linklabel.LinkClicked, AddressOf LinkLabel_LinkClicked
```

Responding to the VB.NET LinkLabel LinkClicked event.

```
Private Sub LinkLabel_LinkClicked(ByVal sender As Object, ByVal e As
LinkLabelLinkClickedEventArgs)
MessageBox.Show("Link Clicked")
End Sub
```

Figure 10.5: LinkLabel Control



Source: http://www.speakcomputers.com/Windows-Forms-Programming/VB/LinkLabel.aspx

Self Assessment

Fill in the blanks:

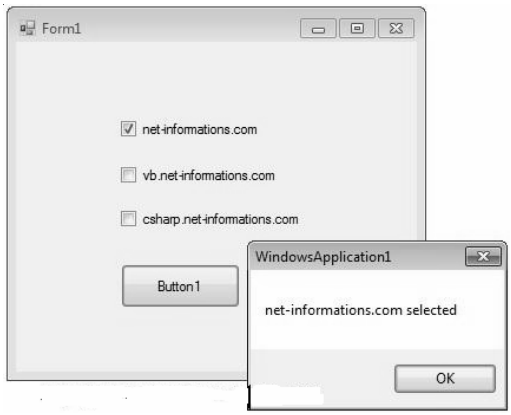
10. A ..... control lets you place descriptive text, where the text does not need to be changed by the user.

11. The ..... control can display single or multiple hyperlinks.

10.6 CheckBox

CheckBoxes allow the user to make multiple selections from a number of options. You can click a checkBox to select it and click it again to deselect it.

Figure 10.6: CheckBox Control



Source: http://www.speakcomputers.com/Windows-Forms-Programming/VB/LinkLabel.aspx

Notes

CheckBoxes come with a caption, which you can set in the Text property.

```
CheckBox1.Text = "Net-informations.com"
```

You can use the CheckBox control ThreeState property to direct the control to return the Checked, Unchecked, and Indeterminate values. You need to set the check box ThreeState property to True to indicate that you want it to support three states.

```
CheckBox1.ThreeState = True
```

To apply the same property settings to multiple CheckBox controls, use the Style property. The following VB.Net program shows how to find a checkbox is selected or not.

Example:

```
Public Class Form1

   Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
      Dim msg As String = ""

      If CheckBox1.Checked = True Then
         msg = "net-informations.com"
      End If

      If CheckBox2.Checked = True Then
         msg = msg & "    vb.net-informations.com"
      End If

      If CheckBox3.Checked = True Then
         msg = msg & "    csharp.net-informations.com"
      End If


      If msg.Length > 0 Then
         MsgBox(msg & " selected ")
      Else
         MsgBox("No checkbox selected")
      End If

      CheckBox1.ThreeState = True
   End Sub

End Class
```

Self Assessment

True or False:

12. CheckBoxes allow the user to make single selection from a number of options.
13. You need to set the checkbox ThreeState property to Tree to indicate that you want to support threestates.

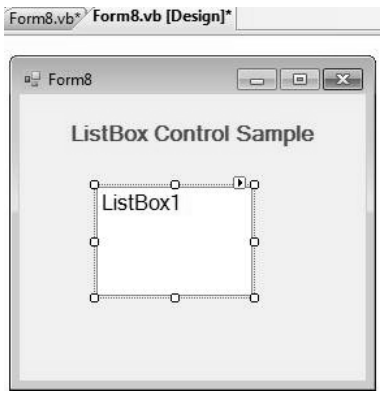
10.7 ListBox

Notes

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.

Drag and drop ListBox from toolbox on the window Form.

Figure 10.7: ListBox Control

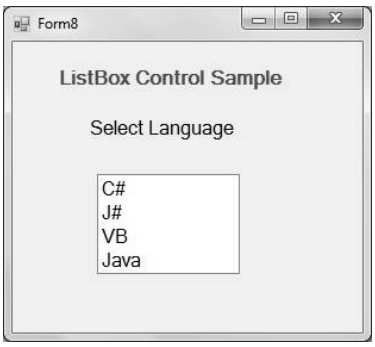


Source: http://www.mindstick.com/Articles/35d84ede-e970-4c74-98db-a64b64d3265f/?ListBox%20Control%20in%20VB.Net

```
Public Class Form8
      Private Sub Form8_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
          'Item is added in ListBox
           ListBox1.Items.Add("C#")
           ListBox1.Items.Add("J#")
           ListBox1.Items.Add("VB")
           ListBox1.Items.Add("Java")
     End Sub

      Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ListBox1.SelectedIndexChanged
          'selected option will show in Label using the selected item property
           Label2.Text = "Selected Language is  " + ListBox1.SelectedItem
     End Sub
End Class
```

In above code Item is added dynamically so all Items in ListBox1 will show at run time.

Figure 10.8: Output of the Code Above

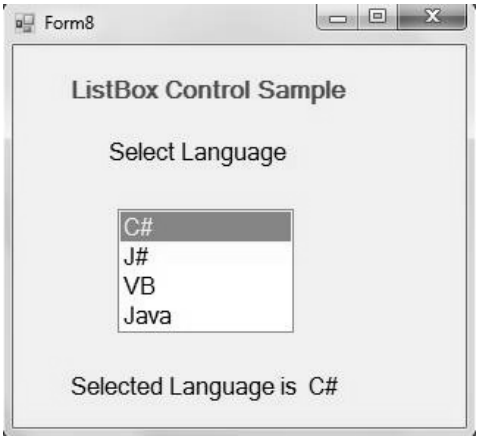


Source: http://www.mindstick.com/Articles/35d84ede-e970-4c74-98db-a64b64d3265f/?ListBox%20Control%20in%20VB.Net

Notes

When you select given option then SelectedIndexChanged event will fire and selected Item will show in the Label.

Figure 10.9: Output of the Code Above



Source: http://www.mindstick.com/Articles/35d84ede-e970-4c74-98db-a64b64d3265f/?ListBox%20Control%20in%20VB.Net

10.7.1 Properties of the ListBox

The properties of a ListBox are summarized in Table 10.7.

Table 10.7: ListBox Properties

S.No.	Event	Description
1.	AllowSelection	Gets a value indicating whether the ListBox currently enables selection of list items.
2.	BorderStyle	Gets or sets the type of border drawn around the list box.
3.	ColumnWidth	Gets of sets the width of columns in a multicolumn list box.
4.	HorizontalExtent	Gets or sets the horizontal scrolling area of a list box.
5.	HorizontalScrollBar	Gets or sets the value indicating whether a horizontal scrollbar is displayed in the list box.
6.	ItemHeight	Gets or sets the height of an item in the list box.
7.	Items	Gets the items of the list box.
8.	MultiColumn	Gets or sets a value indicating whether the list box supports multiple columns.
9.	ScrollAlwaysVisible	Gets or sets a value indicating whether the vertical scroll bar is shown at all times.
10.	SelectedIndex	Gets or sets the zero-based index of the currently selected item in a list box.
11.	SelectedIndices	Gets or sets the zero-based index of the currently selected item in a list box.
12.	SelectedItem	Gets or sets the currently selected item in the list box.
13.	SelectedItems	Get a collection containing the currently selected items in the list box.
14.	SelectedValue	Gets or sets the value of the member property specified by the ValueMember property.

Source: http://www.tutorialspoint.com/vb.net/vb.net_listbox.htm

Notes

Self Assessment

Fill in the blanks:

14. The ..... control displays a list of items from which we can make a selection.
15. When you select given option then ..... event will fire and selected Item will show in the Label.

10.8 ComboBox

The ComboBox control is used to display a drop-down list of various items. It is a combination of a text box in which the user enters an item and a drop-down list from which the user selects an item.

You can populate the list box items either from the properties window or at runtime. To add items to a ListBox, select the ListBox control and get to the properties window, for the properties of this control. Click the ellipses (...) button next to the Items property. This opens the String Collection Editor dialog box, where you can enter the values one at a line.

10.8.1 Properties of the ComboBox

The following are some of the commonly used properties of the ComboBox control:

Table 10.8: ComboBox Properties

S.No.	Property	Description
1.	AllowSelection	Gets a value indicating whether the list enables selection of list items.
2.	AutoCompleteCustomSource	Gets or sets custom System.Collections.Specialized.StringCollection to use when the AutoCompleteSourceproperty is set to CustomSource.
3.	AutoCompleteMode	Gets or sets an option that controls how automatic completion works for the ComboBox.
4.	AutoCompleteSource	Gets or sets a value specifying the source of complete strings used for automatic completion.
5.	DataBindings	Gets the data bindings for the control.
6.	DataManager	Gets the CurrencyManager associated with this control.
7.	DataSource	Gets or sets the data source for this ComboBox.
8.	DropDownHeight	Gets or sets the height in pixels of the drop-down portion of the ComboBox.
9.	DropDownStyle	Gets or sets a value specifying the style of the combo box.
10.	DropDownWidth	Gets or sets the width of the drop-down portion of a combo box.
11.	DroppedDown	Gets or sets a value indicating whether the combo box is displaying its drop-down portion.
12.	FlatStyle	Gets or sets the appearance of the ComboBox.
13.	ItemHeight	Gets or sets the height of an item in the combo box.
14.	Items	Gets an object representing the collection of the item contained in this combo box

Contd...

Notes

15.	MaxDropDownItems	Gets or sets the maximum number of items to be displayed in the drop-down part of the combo box.
16.	MaxLength	Gets or sets the maximum number of characters a user can enter in the editable area of the combo box.

Source: http://www.tutorialspoint.com/vb.net/vb.net_combobox.htm

10.9 TreeView

TreeView control is used to display hierarchical tree like information such as a directory hierarchy. The top level in a tree view are root nodes that can be expanded or collapsed if the nodes have child nodes.

The user can expand the TreeNode by clicking the plus sign (+) button, if one is displayed next to the TreeNode, or you can expand the TreeNode by calling the TreeNode.Expand method. When a parent node is expanded, its child nodes are visible. You can also navigate through tree views with various properties: FirstNode, LastNode, NextNode, PrevNode, NextVisibleNode, PrevVisibleNode.

The fullpath method of treeview control provides the path from root node to the selected node.

```
TreeView1.SelectedNode.FullPath
```

Treenodes can optionally display check boxes. To display the check boxes, set the CheckBoxes property of the TreeView to true.

```
TreeView1.CheckBoxes = True
```

Example:

```
Public Class Form1

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

       Dim tNode As TreeNode
       tNode = TreeView1.Nodes.Add("Websites")

       TreeView1.Nodes(0).Nodes.Add("Net-informations.com")
       TreeView1.Nodes(0).Nodes(0).Nodes.Add("CLR")

       TreeView1.Nodes(0).Nodes.Add("Vb.net-informations.com")
       TreeView1.Nodes(0).Nodes(1).Nodes.Add("String Tutorial")
       TreeView1.Nodes(0).Nodes(1).Nodes.Add("Excel Tutorial")

       TreeView1.Nodes(0).Nodes.Add("Csharp.net-informations.com")
       TreeView1.Nodes(0).Nodes(2).Nodes.Add("ADO.NET")
       TreeView1.Nodes(0).Nodes(2).Nodes(0).Nodes.Add("Dataset")

   End Sub

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

       MsgBox(TreeView1.SelectedNode.FullPath)

   End Sub
End Class
```

Notes

10.9.1 Properties of TreeView

Some of the properties of a TreeView are given below.

Table 10.9: TreeView Properties

S.No.	Property	Description
1.	BackColor	Gets or sets the background color for the control.
2.	BackgroundImage	Gets or set the background image for the TreeView control.
3.	BackgroundImageLayout	Gets or sets the layout of the background image for the TreeView control.
4.	BorderStyle	Gets or sets the border style of the tree view control.
5.	CheckBoxes	Gets or sets a value indicating whether check boxes are displayed next to the tree nodes in the tree view control.
6.	DataBindings	Gets the data bindings for the control.
7.	Font	Gets or sets the font of the text displayed by the control.
8.	FontHeight	Gets or sets the height of the font of the control.
9.	ForeColor	The current foreground color for this control, which is the color the control uses to draw its text.
10.	ItemHeight	Gets or sets the height of each tree nodes in the tree view control.
11.	Nodes	Gets the collection of tree nodes that are assigned to the tree view control.
12.	PathSeparator	Gets or sets the delimiter string that the tree nodes path uses.
13.	RightToLeftLayout	Gets or sets a value that indicates whether the TreeView should be laid out from right-to-left.
14.	Scrollable	Gets or sets a value indicating whether the tree view control displays scroll bars when they are needed.
15.	SelectedImageIndex	Gets or sets the image list index value of the image that is displayed when a tree node is selected.
16.	SelectedImageKey	Gets or sets the key of the default image shown when a TreeNode is in a selected state.
17.	SelectedNode	Gets or sets the tree node that is currently selected in the tree view control.

Source: http://www.tutorialspoint.com/vb.net/vb.net_treeview.htm

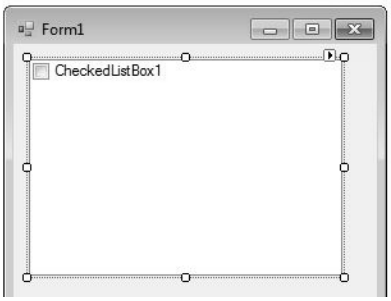
10.10 CheckedListBox

A CheckedListBox control is a ListBox control with CheckBox displayed in the left side where user can select a single or multiple items. We can create a CheckedListBox control using a Forms designer at design-time or using the CheckedListBox class in code at run-time (also known as dynamically).

To create a CheckedListBox control at design-time, you simply drag and drop a CheckedListBox control from Toolbox to a Form in Visual Studio. After you drag and drop a CheckedListBox on a Form, the CheckedListBox looks like Figure 10.10. Once a CheckedListBox is on the Form, you can move it around and resize it using mouse and set its properties and events.

Notes

Figure 10.10: CheckedListBox



Source: http://www.dotnetheaven.com/article/checkedlistbox-in-vb.net

Creating a CheckedListBox control at run-time is merely a work of creating an instance of CheckedListBox class, set its properties and adds CheckedListBox class to the Form controls.

First step to create a dynamic CheckedListBox is to create an instance of CheckedListBox class. The following code snippet creates a CheckedListBox control object.

```
CheckedListBox1 = New CheckedListBox()
```

In the next step, you may set properties of a CheckedListBox control. The following code snippet sets location, width, height, background color, foreground color, Text, Name, and Font properties of a CheckedListBox.

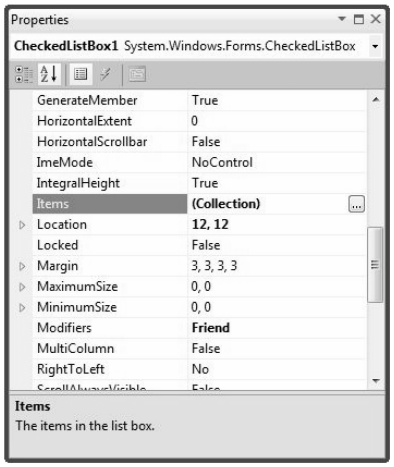
```
CheckedListBox1.Location = New System.Drawing.Point(12, 12)
CheckedListBox1.Name = "CheckedListBox1"
CheckedListBox1.Size = New System.Drawing.Size(245, 169)
CheckedListBox1.BackColor = System.Drawing.Color.Orange
CheckedListBox1.ForeColor = System.Drawing.Color.Black
CheckedListBox1.FormattingEnabled = True
```

Once a CheckedListBox control is ready with its properties, next step is to add the CheckedListBox control to the Form. To do so, we use Form.Controls.Add method. The following code snippet adds a CheckedListBox control to the current Form.

```
Controls.Add(CheckedListBox1)
```

Some of its properties include,

Figure 10.11: CheckedListBox Properties



Source: http://www.dotnetheaven.com/article/checkedlistbox-in-vb.net

Notes

10.11 Panel, GroupBox and PictureBox

We will now discuss each of them individually in detail.

10.11.1 Panel

The Panel control is a container control that is used to host a group of similar child controls. One of the major uses has been found for a Panel control when you have to show and hide a group of controls. Instead of show and hide individual controls, you can simply hide and show a single Panel and all child controls.

We can create a Panel control using the Forms designer at design-time or using the Panel class in code at run-time.

- ***Design-time:*** To create a Panel control at design-time, you simply drag and drop a Panel control from Toolbox to a Form in Visual Studio. Once a Panel is on the Form, you can move it around and resize it using mouse and set its properties and events.
- ***Run-time:*** Creating a Panel control at run-time is merely a work of creating an instance of Panel class, set its properties and adds Panel class to the Form controls.

First step to create a dynamic Panel is to create an instance of Panel class. The following code snippet creates a Panel control object.

```
Dim dynamicPanel As New Panel()
```

In the next step, you may set properties of a Panel control. The following code snippet sets location, size and Name properties of a Panel.

```
dynamicPanel.Location = New System.Drawing.Point(26, 12)
dynamicPanel.Name = "Panel1"
dynamicPanel.Size = New System.Drawing.Size(228, 200)
dynamicPanel.BackColor = Color.LightBlue
```

Once the Panel control is ready with its properties, the next step is to add the Panel to a Form. To do so, we use Form.Controls.Add method that adds Panel control to the Form controls and displays on the Form based on the location and size of the control. The following code snippet adds a Panel control to the current Form.

```
Controls.Add(dynamicPanel)
```

Setting Panel Properties

After you place a Panel control on a Form, the next step is to set its properties.

The easiest way to set properties is from the Properties Window. You can open Properties window by pressing F4 or right click on a control and select Properties menu item.

Adding Controls to a Panel

You can add controls to a Panel by dragging and dropping control to the Panel. We can add controls to a Panel at run-time by using its Add method. The following code snippet creates a Panel, creates a TextBox and a CheckBox and adds these two controls to a Panel.

```
Private Sub CreateButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles CreateButton.Click
          Dim dynamicPanel As New Panel()
           dynamicPanel.Location = New System.Drawing.Point(26, 12)
```

Notes

```
        dynamicPanel.Name = "Panel1"
         dynamicPanel.Size = New System.Drawing.Size(228, 200)
         dynamicPanel.BackColor = Color.LightBlue
        Dim textBox1 As New TextBox()
        textBox1.Location = New Point(10, 10)
        textBox1.Text = "I am a TextBox5"
        textBox1.Size = New Size(200, 30)
        Dim checkBox1 As New CheckBox()
        checkBox1.Location = New Point(10, 50)
        checkBox1.Text = "Check Me"
        checkBox1.Size = New Size(200, 30)
         dynamicPanel.Controls.Add(textBox1)
         dynamicPanel.Controls.Add(checkBox1)
        Controls.Add(dynamicPanel)
    End Sub
```

Show and Hide a Panel

Applications where you want to show and hide a group of controls on a Form based on some condition. That is where a Panel is useful. Instead of show and hide individual controls, we can group controls that we want to show and hide and place them on two different Panels and show and hide the Panels. To show and hide a Panel, we use Visible property.

```
dynamicPanel.Visible = False
```

10.11.2 GroupBox Control

GroupBox is a container of other control. It displays a frame around a group of controls. When you move the GroupBox control, all of its contained will also move.

Drag and drop GroupBox control from toolbox on the window Form. When you drag and drop group box, it has a border by default, and its caption is set to the name of the control. You can change its caption. GroupBox is a container of other control.

GroupBox Properties

The following are the properties of GroupBox:

- ***AutoSize:*** Gets or sets a value that indicates whether the GroupBox resizes based on its contents.
- ***BackColor:*** Gets or sets the background color for the control.
- ***ForeColor:*** Change ForeColor of all control in side GroupBox

You can change ForeColor of all control in side GroupBox at a time.

Code should be written inside Form Load so that when Form run then code (for changing Fore Color) will be executed.

Example:

```
Private Sub Form24_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
          'Change Fore color of all control in side GroupBox
          GroupBox1.ForeColor = Color.Blue
 End Sub
```

Forecolor of all control in side GroupBox will be changed when Application run.

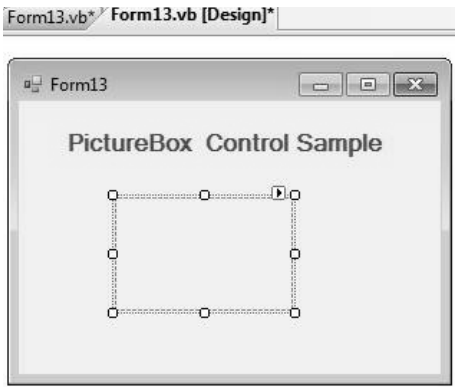
Notes

10.11.3 PictureBox Control

PictureBox control is used to display image. The images displayed can be any format like Bitmap, JPEG, and GIF, PNG or any other image format files. The PictureBox control is based on the Control class.

Drag and drop the PictureBox control from toolbox on the Window Form.

Figure 10.12: PictureBox Properties



Source: http://www.mindstick.com/Articles/510a9eaa-1901-4c19-b2e1-e21f0176a5e3/?PictureBox%20Control%20in%20VB.Net

```
Code:
Public Class Form13
Private Sub Form13_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
   'show image in PictureBox
  PictureBox1.Image = Image.FromFile("C:\Pictures\Sample Pictures\image1.png")
     End Sub
End Class
```

FromFile method is used to find the image.

Run the Project

Image will show in PictureBox when application run.

Figure 10.13: PictureBox

Source: http://www.mindstick.com/Articles/510a9eaa-1901-4c19-b2e1-e21f0176a5e3/?PictureBox%20Control%20in%20VB.Net

Notes

Size Mode property of PictureBox: Set the Image size mode by default it set to Normal.

```
Private Sub Form13_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
          'set sizemode to CenterImage
           PictureBox1.SizeMode = PictureBoxSizeMode.CenterImage
 End Sub
```

Centre image will show in PictureBox.

Figure 10.14: Output of the Code Above

Source: http://www.mindstick.com/Articles/510a9eaa-1901-4c19-b2e1-e21f0176a5e3/?PictureBox%20Control%20in%20VB.Net

10.12 ToolTip and Error Provider Component

Now we will discuss the two very useful controls in VB.NET.

10.12.1 ToolTip

ToolTip makes interfaces more intuitive. We add the ToolTip control to a Windows Forms application, which provides useful contextual hints to the client. We use the Visual Studio designer to easily add ToolTips.

Figure 10.15: ToolTip

Source: http://www.dotnetperls.com/tooltip

Setting a ToolTip

To set the tooltip we use the SetToolTip( ) method.

Example:

```
Me.tipExample1.SetToolTip(Me.lblInput, "This is a label for the input
field")
```

Notes

```
Me.tipExample1.SetToolTip(Me.txtInput, "Please enter your first name")
Me.tipExample1.SetToolTip(Me.btnExecute, "Clicking this button generates a
message for you")
```

or to create a new line in the tooltip:

```
Me.tipExample1.SetToolTip( Me.Button1, "Foo" & ControlChars.NewLine & "Bar")
```

10.12.2 ErrorProvider Component

Suppose you want to give your users a clue that their input is wrong somehow, but you don't want to display a whole popup window like a MessageBox. Windows Forms gives you the answer with the **ErrorProvider** component. In VB.old, a MessageBox was about all you had. ErrorProvider is way more cool.

There are three good reasons for using ErrorProvider instead of popping up a MessageBox.

1. A MessageBox interrupts the flow of the program and you have to click a button to get out of it again.
2. You can't show visually where the error actually is.
3. You can't show multiple errors.

ErrorProvider has a lot of flexibility. For example, when the DataSource property is set, the ErrorProvider component can display error messages for a dataset. To demonstrated some of the flexibility in the example code here, we changed the Icon property from the default.

To use ErrorProvider, drag it from the ToolBox onto your form. It shows up in the component tray at the bottom of the designer window since it doesn't have a visual presence on the form. (In other words, it's like the Timer control.) Then, when you want to display the error icon, just call the SetError method. When the SetError method is called, ErrorProvider displays an icon (the default is the red exclamation point in a circle) and will also display ToolTip text for the error.

Displaying an Error

Here's an example of code that uses ErrorProvider. This code displays four different icons that insult the user's choice of fonts (no matter what is selected):

```
Private Sub Form1_Load( ...
   For Each ff In _
        System.Drawing.FontFamily.Families
        availableFonts.Items.Add(ff.Name)
   Next
End Sub
Private Sub availableFonts_SelectedIndexChanged( ...
    DisplayErrorProvider(sender)
   errCount += 1
End Sub
Private Sub DisplayErrorProvider(ByVal sender As Object)
    Select Case errCount
      Case 0
           ErrorProvider1.SetError( _
          sender, "What a dumb font!" & _
          vbCrLf & "I don't know whether" & _
```

Notes

```
                vbCrLf & "to laugh or cry.")
                ErrorProvider1.Icon = _
                New Icon("C:\f1.ico")
            Case 1
                ErrorProvider1.SetError( _
                sender, "You did it again!" & _
                vbCrLf & "I'll laugh it off this time!")
                ErrorProvider1.Icon = _
                New Icon("C:\f2.ico")
            Case 2
                ErrorProvider1.SetError( _
                sender, "You gotta cut this out!" & _
                vbCrLf & "I'm getting mad now!")
                ErrorProvider1.Icon = _
                New Icon("C:\f3.ico")
            Case Else
                ErrorProvider1.SetError( _
                sender, "This is a lost cause!" & _
                vbCrLf & "You're hopeless.")
                ErrorProvider1.Icon = _
                New Icon("C:\f4.ico")
        End Select
    End Sub
```

10.13 StatusBar

StatusBar control is not available in Toolbox of Visual Studio 2010. StatusStrip control replaces StatusBar in Visual Studio 2010. But for backward compatibility support, StatusBar class is available in Windows Forms.

A StatusBar control is a combination of StatusBar panels where each panel can be used to display different information. For example, one panel can display current application status and other can display date and other information and so on. A typical StatusBar sits at the bottom of a form.

```
StatusBar class represents a StatusBar.
Dim mainStatusBar As New StatusBar()
```

A StatusBar is a combination of StatusBar panels. StatusBarPanel class represents a StatusBar panel. The following code snippet creates two panels and adds them to the StatusBar.

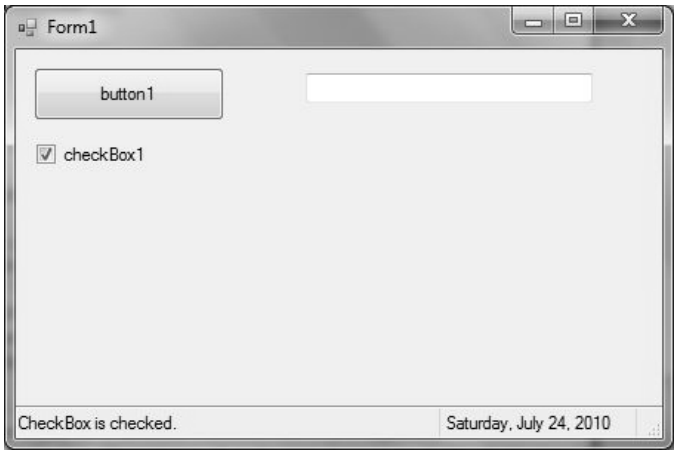
```
Dim statusPanel As New StatusBarPanel()
Dim datetimePanel As New StatusBarPanel()

statusPanel.BorderStyle = StatusBarPanelBorderStyle.Sunken
statusPanel.Text = "Application started. No action yet."
statusPanel.ToolTipText = "Last Activity"
statusPanel.AutoSize = StatusBarPanelAutoSize.Spring
mainStatusBar.Panels.Add(statusPanel)
datetimePanel.BorderStyle = StatusBarPanelBorderStyle.Raised
datetimePanel.ToolTipText = "DateTime: " + System.DateTime.Today.ToString()
datetimePanel.Text = System.DateTime.Today.ToLongDateString()
datetimePanel.AutoSize = StatusBarPanelAutoSize.Contents
mainStatusBar.Panels.Add(datetimePanel)
```

Notes

```
Now, make sure ShowPanels property is true.
mainStatusBar.ShowPanels = True

In the end, we add StatusBar to the Form.

Controls.Add(mainStatusBar)
```

Now let's create a Windows Forms application with a few controls on it. We are going to show current activity and date on the status bar. The Form looks like following.

Figure 10.16: Status Bar



Source: http://www.dotnetheaven.com/article/statusbar-control-in-vb-.net

10.14 RadioButton

A radio button or option button is a type of graphical user interface element that allows the user to choose only one of a predefined set of options. When a user clicks on a radio button, it becomes checked, and all other radio buttons with same group become unchecked. The radio button and the check box are used for different functions. Use a radio button when you want the user to choose only one option. When you want the user to choose all appropriate options, use a check box. Like check boxes, radio buttons support a Checked property that indicates whether the radio button is selected.

Example:

```
Public Class Form1

      Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
          RadioButton1.Checked = True
     End Sub

      Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
          If RadioButton1.Checked = True Then
              MsgBox("You are selected Red !! ")
             Exit Sub
          ElseIf RadioButton2.Checked = True Then
```

Notes

```
                MsgBox("You are selected Blue !! ")
               Exit Sub
          Else
                MsgBox("You are selected Green !! ")
               Exit Sub
          End If
     End Sub

End Class
```

Case Study

.NET Memory Leak: The Event Handlers that Made the Memory Baloon

It always feels like issues come in clusters. One week we get tones of cache related cases, next week everyone is stuck in some lock and so on. Lately I have had a number of issues where there were memory leaks related to event handlers.

Problem Description

We have a pretty easy to repro memory leak, even with very few users memory grows at a high rate and the memory is never released.

The usual suspects are eliminated, i.e. we know that the application doesn't cache much or store much or anything in session scope, and from performance monitor we can see that as private bytes grow, most or all of this growth is in the .net heaps (if private bytes grow by 100 MB, # bytes in all heaps also grow by about 100 MB).

Gathering Data

Since we can tell with !dumpheap – state exactly what objects are on the managed heap, the natural path to take here is to take two dumps as the memory grows and compare and see what types of objects increase the most.

Preferably to make the problem more visible, a stress test is nice so that the leaking objects stand out more. For example if we leak one dataset per pageview, and you see one dataset in the dump it doesn't really stand out, but perhaps 10.000 would.

When you take the first dump (adplus –hang –pn w3wp.exe), do this after loading all relevant pages at least once so that all assemblies and dlls are loaded. After this, stress the application, and get another dump.

Debugging

As usual I have simplified the scenario a bit, but the debugging techniques stay the same.

I have a simple page, hardly any code in it at all, stressed it for just 14 seconds using ACT (got ~7000 page hits) and the memory grew by an amazing 64 MB. Of course I would recommend that you stress more thank 14 seconds, but in this case it was enough to give me a pretty good leak.

We can see how much memory was used by the process in the two instances by just looking at the size of the dump files. The first dump was at 65 MB and the second at 129 MB.

!eeheap –gc gives us information about the size of the managed heap along with some segment information etc. The first dump showed that we used 5 MB of .net heap, and the

Contd...

Notes

second shows that this has grown to 73,004,928 bytes (~69 MB), so an increase of 64 MB which matches the problem statement, that "all" our memory increase is on the .net heap.

A Short Discussion on Event Handlers

An event is really a linked list of event handlers as you can see from dumping out the eventhandlers above (each node contains the target, method and a prev pointer).

When you do += new EventHandler on the event it doesn't really matter if this particular function has been added as a listener before, it will get added once per +=.

When the event is raised we go through the linked list, item by item and call all the methods (event handlers) added to this list, this is why the event handlers are still called even when the pages are no longer running as long as they are alive (rooted), and they will be alive as long as they are hooked up. So they will get called until the eventhandler is unhooked with a -= new EventHandler.

In VB.Net you hook up events with AddHandler or with the Handles keyword, both hook up the events the same way. The equivalent to the -= new EventHandler… in vb.net, is RemoveHandler.

The moral of the story here is: be very careful if you are hooking up event handlers to objects that have a longer lifespan than the page.

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://blogs.msdn.com/b/tess/archive/2006/01/23/net-memory-leak-case-study-the-event-handlers-that-made-the-memory-baloon.aspx

Self Assessment

Fill in the blanks:

16. A ..... control is a ListBox control with CheckBox displayed in the left side where user can select a single or multiple items.
17. A ..... button is a type of graphical user interface element that allows the user to choose only one of a predefined set of options.

10.15 Summary

- Windows Forms is a framework located in the System.Windows.Forms.dll assembly.
- The Control Class defines the base class for controls.
- Tab Index property is set so that we can move focus from one control to other.
- Click event is the default event of a button.
- All Visual Basic .NET constants map to the System.Windows.Forms namespace unless otherwise noted.
- A RichTextBox control is an advanced text box that provides text editing and advanced formatting features including loading rich text format (RTF) files.

Notes

- Creating a RichTextBox control at run-time is merely a work of creating an instance of RichTextBox class.
- A Label control lets you place descriptive text , where the text does not need to be changed by the user.
- The Label class is defined in the System.Windows.Forms namespace.
- The LinkLabel control can display single or multiple hyperlinks.
- CheckBoxes allow the user to make multiple selections from a number of options.
- To apply the same property settings to multiple CheckBox controls, use the Style property.
- The ListBox control displays a list of items from which we can make a selection.
- When you select given option then SelectedIndexChanged event will fire and selected Item will show in the Label.
- The ComboBox control is used to display a drop-down list of various items.
- TreeView control is used to display hierarchical tree like information such as a directory hierarchy.
- A CheckedListBox control is a ListBox control with CheckBox displayed in the left side where user can select a single or multiple items.
- A radio button or option button is a type of graphical user interface element that allows the user to choose only one of a predefined set of options.

10.16 Keywords

CheckBoxes: They allow the user to make multiple selections from a number of options.

Control Class: It defines the base class for controls.

Label control: It lets you place descriptive text , where the text does not need to be changed by the user.

LinkLabel control: It can display single or multiple hyperlinks.

ListBox control: It displays a list of items from which we can make a selection.

RichTextBox control: It is an advanced text box that provides text editing and advanced formatting features including loading rich text format (RTF) files.

Tab Index property: It is set so that we can move focus from one control to other.

Windows Forms: It is a framework located in the System.Windows.Forms.dll assembly.

10.17 Review Questions

1. What are Windows Controls?
2. Explain the Control Tab Order concept.
3. Write a short note on Button Control.
4. Which is the default Button Event?
5. Explain the TextBox Control and list its properties.
6. Differentiate between TextBox and RichTextBox.

Notes

7. What are the major differences between Label and Link Label?
8. Explain the use of TreeView control.
9. Write the code to set the ToolTip.
10. What are radio buttons? How are they different from checkboxes?

Answers: Self Assessment

1.	True	2.	False
3.	False	4.	Select
5.	Click	6.	System.Windows.Forms
7.	Left, Right, Center	8.	True
9.	False	10.	Label
11.	LinkLabel	12.	False
13.	True	14.	ListBox
15.	SelectedIndexChanged	16.	CheckedListBox
17.	Radio		

10.18 Further Readings

Books

Beginning Vb.Net 2003, Willis.

Object-oriented Programming with Visual Basic.Net, Hamilton.

Programming Visual Basic .NET, J. Liberty.

Visual Basic.NET Black Book, Steven Holzner.

Online links

http://www.dotnetheaven.com/article/groupbox-control-in-vb.net

http://www.mindstick.com/Articles/cae0875c-08cb-4094-aeda-676bcb5b450b/?GroupBox%20Control%20in%20VB.Net

http://www.java2s.com/Code/VBAPI/System.Windows.Forms/GroupBoxControlsAddRange.htm

http://www.tutorialspoint.com/vb.net/vb.net_basic_controls.htm

Notes

Unit 11: Common Dialog Boxes

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain common dialog boxes
- Discuss the common dialog control
- Define save file dialog box
- Understand the color dialog box
- Describe the possible dialog result values

Notes

Introduction

Standalone applications typically have a main window that both displays the main data over which the application operates and exposes the functionality to process that data through User Interface (UI) mechanisms like menu bars, tool bars, and status bars. A non-trivial application may also display additional windows to do the following:

- Display specific information to users.
- Gather information from users.
- Both display and gather information.

These types of windows are known as dialog boxes, and there are two types: modal and modeless.

A modal dialog box is displayed by a function when the function needs additional data from a user to continue. Because the function depends on the modal dialog box to gather data, the modal dialog box also prevents a user from activating other windows in the application while it remains open. In most cases, a modal dialog box allows a user to signal when they have finished with the modal dialog box by pressing either an OK or Cancel button. Pressing the OK button indicates that a user has entered data and wants the function to continue processing with that data. Pressing the Cancel button indicates that a user wants to stop the function from executing altogether. The most common examples of modal dialog boxes are shown to open, save, and print data.

A modeless dialog box, on the other hand, does not prevent a user from activating other windows while it is open. For example, if a user wants to find occurrences of a particular word in a document, a main window will often open a dialog box to ask a user what word they are looking for. Since finding a word doesn't prevent a user from editing the document, however, the dialog box doesn't need to be modal. A modeless dialog box at least provides a Close button to close the dialog box, and may provide additional buttons to execute specific functions, such as a Find Next button to find the next word that matches the find criteria of a word search.

Windows Presentation Foundation (WPF) allows you to create several types of dialog boxes, including message boxes, common dialog boxes, and custom dialog boxes.

11.1 Common Dialog Boxes

There are many built-in dialog boxes to be used in Windows forms for various tasks like opening and saving files, printing a page, providing choices for colors, fonts, page set up etc. to the user of an application. These built-in dialog boxes reduce the developer's time and work load. All of these dialog box control classes inherit from the CommonDialog class and override the *RunDialog()* function of the base class to create the specific dialog box. The RunDialog() function is automatically invoked when a user of a dialog box calls its *ShowDialog()* function. The ShowDialog method is used to display all the dialog box controls at run time. It returns a value of the type of DialogResult enumeration. The values of DialogResult enumeration are:

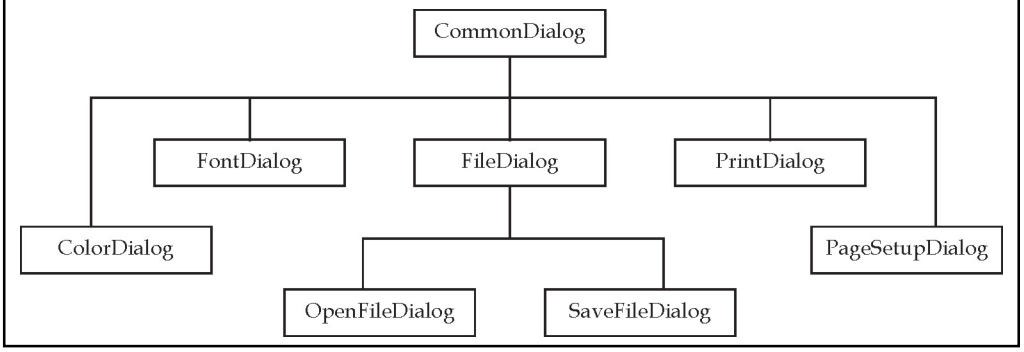
- ***Abort:*** returns DialogResult.Abort value, when user clicks an Abort button.
- ***Cancel:*** returns DialogResult.Cancel, when user clicks a Cancel button.
- ***Ignore:*** returns DialogResult.Ignore, when user clicks an Ignore button.
- ***No:*** returns DialogResult.No, when user clicks a No button.
- ***None:*** returns nothing and the dialog box continues running.
- ***OK:*** returns DialogResult.OK, when user clicks an OK button.

Notes

- ***Retry:*** returns DialogResult.Retry, when user clicks an Retry button.
- ***Yes:*** returns DialogResult.Yes, when user clicks an Yes button.

The following diagram shows the common dialog class inheritance:

Figure 11.1: Common Dialog Boxes



Source: http://www.tutorialspoint.com/vb.net/vb.net_dialog_boxes.htm

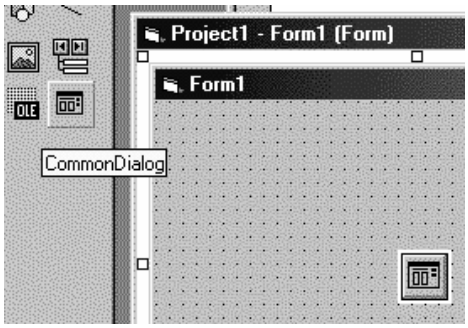
All these above mentioned classes have corresponding controls that could be added from the Toolbox during design time. You can include relevant functionality of these classes to your application, either by instantiating the class programmatically or by using relevant controls. When you double click any of the dialog controls in the toolbox or drag the control onto the form, it appears in the Component tray at the bottom of the Windows Forms Designer, they do not directly show up on the form.

11.1.1 Using a Common Dialog Box

To use the Common Dialog Control, you need to place it on your form in the same way as you would place other controls on your form–by double-clicking on the control's tool in the toolbox. You do not need to worry about the size of the control or its position on the form, as it will not be visible when your project is executing. You only need to have one common dialog control on your form, regardless of how many different types of dialog boxes you want to use.

Your form will look like this when the Common Dialog Control has been added to it:

Figure 11.2: Common Dialog Box



Source: http://toolboxes.flexiblelearning.net.au/demosites/series2/210v2/reslib/01/uid_advcontrolvb2.html

11.1.2 Common Dialog Box Library

The Common Dialog Box Library contains a set of dialog boxes for performing common application tasks, such as opening files, choosing color values, and printing documents.

Notes

The common dialog boxes allow you to implement a consistent approach to your application's user interface. This reduces the amount of effort that users spend in learning user interface behavior for your application.

Note Starting with Windows Vista, the **Open and Save As** common dialog boxes have been superseded by the Common Item Dialog. We recommended that you use the Common Item Dialog API instead of these dialog boxes from the Common Dialog Box Library.

The Common Dialog Box Library provides a creation function and a structure for each type of common dialog box. To use a common dialog box in its simplest form, you call its creation function and specify a pointer to a structure that contains initial values and option flags. After initializing the dialog box, the dialog box procedure uses the structure to return information about the user input. You can also customize a common dialog box to suit the needs of your application.

Table 11.1 provides a brief description of the different types of common dialog boxes, and shows the function and structure used with each type.

Table 11.1: Types of a Dialog Box

Color	Displays available colors and optionally lets the user create custom colors. The user can select a basic or custom color. Use the ChooseColor function and CHOOSECOLOR structure.
Find	Displays a dialog box in which the user can type the string to find. The user can also specify search options, such as the search direction and whether the search is case sensitive. Use the FindText function and FINDREPLACE structure.
Font	Displays a dialog box in which the user can select a font family and associated font style, point sizes, and other font attributes such as font color, underline, or strikethrough. Use the ChooseFont function and CHOOSEFONT structure.
Open	Displays a dialog box in which the user can type or select the name of a file or shell name-space object to open. The dialog box includes lists of drives, directories, and shell name-space extensions that enable the user to browse the shell name space. It also includes a list of file name extensions that enables the user to filter the file names displayed. Use the GetOpenFileName function and OPENFILENAME structure.
Page Setup	Displays the current page configuration. The user can select page configuration options, such as paper orientation, size, source, and margins. Use the PageSetupDlg function and PAGESETUPDLG structure
Print	Displays information about the installed printer and its configuration. The user can select print job options, such as the range of pages to print and the number of copies, and start the printing process. Use the PrintDlg function and PRINTDLG structure. For more information, see Print Dialog Box. To display a Print property sheet rather than a Print dialog box, use the PrintDlgEx function with the PRINTDLGEX structure. The General page of the property sheet is similar to the Print dialog box. The property sheet can have additional application-specific and driver-specific property pages following the General page.
Replace	Displays a dialog box in which the user can type the string to find and the replacement string. The user can specify search options, such as whether the search is case sensitive, and replacement options, such as the scope of replacement. Use the ReplaceText function and FINDREPLACE structure.
Save As	Displays a dialog box in which the user can type or select the name with which to save a file or shell name-space object. The dialog box includes lists of drives, directories, and shell name-space extensions that enable the user to browse the shell name space. It also includes a list of file name extensions that enables the user to filter the file names displayed. Use the GetSaveFileName function and OPENFILENAME structure.

Source: http://msdn.microsoft.com/en-us/library/windows/desktop/ms646954%28v=vs.85%29.aspx

Notes

11.1.3 Common Dialog Control

To display any of the common dialog boxes from within your application, you must first add an instance of the appropriate control to your project. Then you must set some basic properties of the control through the Properties window. Most applications set the control's properties from within the code because common dialogs interact closely with the application. When you call the Color common dialog, for example, you should preselect a color from within your application and make it the default selection on the control. When prompting the user for the color of the text, the default selection should be the current setting of the control's ForeColor property. Likewise, the Save dialog box must suggest a filename when it first pops up (or the file's extension, at least).

To display a common dialog box from within your code, you simply call the control's ShowDialog method, which is common for all controls. Note that all common dialog controls can be displayed only modally and they don't expose a Show method. As soon as you call the ShowDialog method, the corresponding dialog box appears onscreen, and the execution of the program is suspended until the box is closed. Using the Open, Save, and FolderBrowser dialog boxes, users can traverse the entire structure of their drives and locate the desired filename or folder. When the user clicks the Open or Save button, the dialog box closes and the program's execution resumes. The code should read the name of the file selected by the user through the FileName property and use it to open the file or store the current document there. The folder selected in the FolderBrowserDialog control is returned to the application through the SelectedPath property.

Example: Here is the sequence of statements used to invoke the Open common dialog and retrieve the selected filename:

```
If OpenFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
fileName = OpenFileDialog1.FileName
' Statements to open the selected file
End If
```

The ShowDialog method returns a value indicating how the dialog box was closed. You should read this value from within your code and ignore the settings of the dialog box if the operation was cancelled. The variable fileName in the preceding code segment is the full pathname of the file selected by the user.

Example: You can also set the FileName property to a filename, which will be displayed when the Open dialog box is first opened:

```
OpenFileDialog1.FileName = "C:\WorkFiles\Documents\Document1.doc"
If OpenFileDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
fileName = OpenFileDialog1.FileName
' Statements to open the selected file
End If
```

Similarly, you can invoke the Color dialog box and read the value of the selected color by using the following statements:

```
ColorDialog1.Color = TextBox1.BackColor
If ColorDialog1.ShowDialog = DialogResult.OK Then
TextBox1.BackColor = ColorDialog1.Color
End If
```

The ShowDialog method is common to all controls. The Title property is also common to all controls and it's the string displayed in the title bar of the dialog box. The default title is the

Notes

name of the dialog box (for example, Open, Color, and so on), but you can adjust it from within your code with a statement such as the following:

```
ColorDialog1.Title = "Select Drawing Color"
```

Self Assessment

Fill in the blanks:

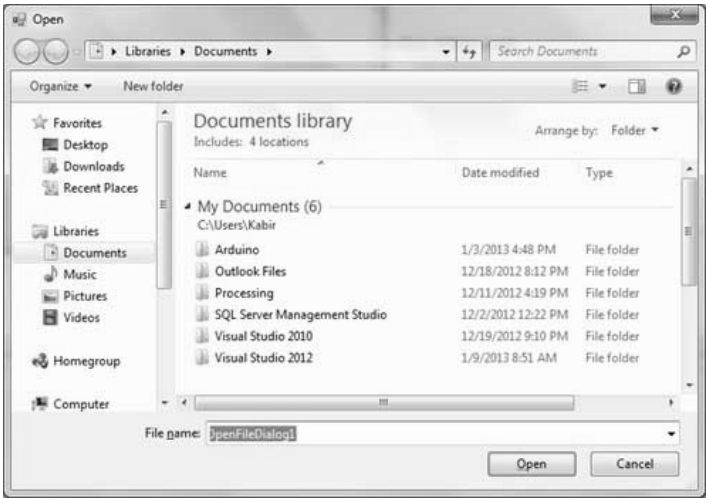
1. A ..... dialog box is displayed by a function when the function needs additional data from a user to continue.
2. The dialog box control classes inherit from the CommonDialog class and override the ..... function of the base class.
3. The ..... method is used to display all the dialog box controls at run time.
4. The ..... Library provides a creation function and a structure for each type of common dialog box.

11.2 OpenFileDialog

The OpenFileDialog control prompts the user to open a file and allows the user to select a file to open. The user can check if the file exists and then open it. The OpenFileDialog control class inherits from the abstract class FileDialog. If the ShowReadOnly property is set to True, then a read-only check box appears in the dialog box. You can also set the ReadOnlyChecked property to True, so that the read-only check box appears checked.

Following is the OpenFileDialog box:

Figure 11.3: OpenFileDialog



Source: http://www.tutorialspoint.com/vb.net/vb.net_openfile_dialog.htm

11.2.1 OpenFileDialog Box Creation

To create an OpenFileDialog box:

```
Imports System
Imports System.Collections
Imports System.ComponentModel
```

Notes

```
Imports System.Windows.Forms
Imports System.Data
Imports System.Configuration
Imports System.Resources
Imports System.Drawing
Imports System.Drawing.Drawing2D
Imports System.IO
Imports System.Drawing.Printing


Public Class MainClass
     Shared Sub Main()
          'Declare a OpenFileDialog object
          Dim objOpenFileDialog As New OpenFileDialog

          'Set the Open dialog properties
          With objOpenFileDialog
                .Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*"
              .FilterIndex = 1
               .Title = "Demo Open File Dialog"
         End With

          'Show the Open dialog and if the user clicks the Open button,
         'load the file
           If objOpenFileDialog.ShowDialog = Windows.Forms.DialogResult.OK
Then
              Dim allText As String
             Try
                   'Read the contents of the file
                     allText = My.Computer.FileSystem.ReadAllText( _
                        objOpenFileDialog.FileName)
                    'Display the file contents in the TextBox
                     System.Console.WriteLine(allText)
               Catch fileException As Exception
                   Throw fileException
             End Try
         End If

         'Clean up
           objOpenFileDialog.Dispose()
          objOpenFileDialog = Nothing
     End Sub

End Class
```

11.2.2 Characteristics of an OpenFileDialog Box

A characteristic of dialog boxes is that they provide an OK and a Cancel button. The OK button tells the application that you're finished using the dialog box, and the application can process the information in it. The Cancel button tells the application that it should ignore the information in the dialog box and cancel the current operation. As you will see, dialog boxes allow you to quickly find out which buttons were clicked to close them, so that your application can take a

Notes

different action in each case. In short, the difference between forms and dialog boxes is artificial. If it were really important to distinguish between the two, they'd be implemented as two different objects–but they're the same object. So, without any further introduction, let's look at how to create and use dialog boxes. To create a dialog box, start with a Windows form, set its FormBorderStyle property to Fixed-Dialog, and set the ControlBox, MinimizeBox, and MaximizeBox properties to False. Then add the necessary controls on the form and code the appropriate events, as you would do with a regular Windows form.

Self Assessment

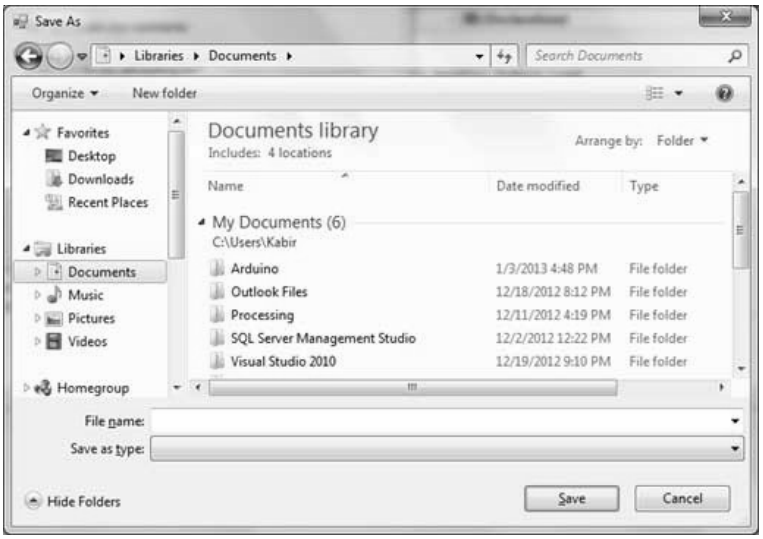
True or False:

5. The OpenFileDialog control class inherits from the abstract class FileDialog
6. A characteristic of dialog boxes is that they provide an Save and a Cancel button

11.3 SaveFileDialog

The SaveFileDialog control prompts the user to select a location for saving a file and allows the user to specify the name of the file to save data. The SaveFileDialog control class inherits from the abstract class FileDialog.

Figure 11.4: SaveFileDialog



Source: http://www.tutorialspoint.com/vb.net/vb.net_savefile_dialog.htm

11.3.1 Creating a SaveFileDialog

To make a Save As dialog box available to your .NET Framework application, on the Toolbox, you can click the SaveFileDialog button and click the form. To programmatically provide this dialog box, you can declare a **SaveFileDialog** variable and initialize it using the class default constructor as follows:

```
Dim sfd As SaveFileDialog

Public Sub New()
        MyBase.New()
```

Notes

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

sfd = New SaveFileDialog

End Sub
```

The SaveFileDialog class inherits from the FileDialog class from which it gets most of its characteristics and behaviors.

Self Assessment

Fill in the blanks:

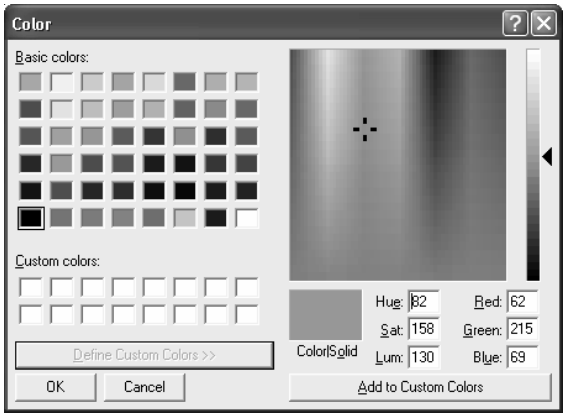
7. The ..... control prompts the user to select a location for saving a file and allows the user to specify the name of the file to save data.
8. The SaveFileDialog control class inherits from the abstract class ..... .

11.4 Color Dialog Box

Displays a modal dialog box that allows the user to choose a specific color value. The user can choose a color from either a set of basic or custom color palettes. Alternatively, the user can generate a color value by modifying the RGB or hue, saturation, luminosity (HSL) color values of the dialog box user interface. The **Color** dialog box returns the RGB value of the color selected by the user.

You create and display **a Color** dialog box by initializing **a CHOOSECOLOR** structure and passing the structure to the **ChooseColor** function. By setting different parameter values for the **CHOOSECOLOR** structure, you can affect how the Color dialog box appears. For example, you can display either a full or partial user interface version of the dialog box.

Figure 11.5: Color Dialog Box



Source: http://msdn.microsoft.com/en-us/library/windows/desktop/ms646375%28v=vs.85%29.aspx

11.4.1 Making a Color Dialog Box Available

The user can choose a color from either a set of basic or custom color palettes. You can invite a color dialog box by calling ShowDialog() method.

```
Dim dlg As New ColorDialog
dlg.ShowDialog()
```

Notes

The Color dialog box has a full version and a partial version of the user interface. The full version includes the basic controls and has additional controls that allow the user to create custom colors. The partial version has controls that display the basic and custom color palettes from which the user can select a color value. The system stores internal colors as 32-bit RGB values that have the following hexadecimal form: 0x00bbggrr.

The following Vb.Net program invites a color dialog box and retrieve the selected color to a string.

```
Public Class Form1
   Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

       Dim dlg As New ColorDialog
       dlg.ShowDialog()

       If dlg.ShowDialog = Windows.Forms.DialogResult.OK Then
           Dim str As String
           str = dlg.Color.Name
           MsgBox(str)
       End If

   End Sub
End Class
```

Self Assessment

True or False:

9. The Color Dialog Box displays a modeless dialog box that allows the user to choose a specific color.
10. You create and display a Color dialog box by initializing a CHOOSECOLOR structure and passing the structure to the ChooseColor function.

11.5 MessageBox Class

Displays a message box that can contain text, buttons, and symbols that inform and instruct the user. Any public static (Shared in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe. You cannot create a new instance of the MessageBox class. To display a message box, call the static (Shared in Visual Basic) method MessageBox.Show. The title, message, buttons, and icons displayed in the message box are determined by parameters that you pass to this method.

Example:

```
Protected Sub button1_Click(sender As Object, e As System.EventArgs)
     If textBox1.Text = "" Then
           MessageBox.Show("You must enter a name.", "Name Entry Error", _
                MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
    Else
          ' Code to act on the data entered would go here.
    End If
End Sub
```

Notes

11.5.1 MessageBox() Function

A message box is a special dialog box used to display a piece of information to the user. As opposed to a regular form, the user cannot type anything in the dialog box. To support message boxes, the Visual Basic language provides a function named MsgBox. To support message boxes, the .NET Framework provides a class named.

To display a simple message box, you can use the MsgBox() function with the following formula:

```
MsgBox(Message)
Inside the parentheses, pass a string.
```

Example:

```
Private Sub btnMessage_Click(ByVal sender As System.Object, _
                                   ByVal e As System.EventArgs) _
                                   Handles btnMessage.Click
          MsgBox("Welcome to Microsoft Visual Basic")
End Sub

Output:
```

Figure 11.6: Output of the Above

Besides displaying a message, a message box can be used to let the user make a decision by clicking a button and, depending on the button the user would have clicked, the message box would return a value. To be able to return a value, the MsgBox() function is declared as follows:

```
Public Shared Function MsgBox ( _
   Prompt As Object, _
   <OptionalAttribute> Optional Buttons As MsgBoxStyle = MsgBoxStyle.OkOnly,
_
   <OptionalAttribute> Optional Title As Object = Nothing _
) As MsgBoxResult
```

The value returned by a message box corresponds to a button the user would have clicked (on the message box). The return value of the MsgBox() function is based on the **MsgBoxResult** enumeration. The buttons and the returned values are as follows:

Table 11.2: Types of Button

If the User Clicks	Button Caption	Integral Value
OK	OK	1
Cancel	Cancel	2
Abort	Abort	3
Retry	Retry	4
Ignore	Ignore	5
Yes	Yes	6
No	No	7

Source: http://www.functionx.com/vb/functions/msgbox.htm

Notes

11.5.2 Input Box() Function

Displays a prompt in a dialog box, waits for the user to input text or click a button, and then returns a string containing the contents of the text box.

```
Public Function InputBox( _
    ByVal Prompt As String, _
    Optional ByVal Title As String = "", _
    Optional ByVal DefaultResponse As String = "", _
    Optional ByVal XPos As Integer = -1, _
    Optional ByVal YPos As Integer = -1 _
) As String
```

Prompt - Required. **String** expression displayed as the message in the dialog box. The maximum length of *Prompt* is approximately 1024 characters, depending on the width of the characters used. If *Prompt* consists of more than one line, you can separate the lines using a carriage return character (**Chr(**13**)**), a linefeed character (**Chr(**10**)**), or a carriage return–linefeed character combination (**Chr(**13**)** & **Chr(**10**)**) between each line.

Title - Optional. **String** expression displayed in the title bar of the dialog box. If you omit *Title*, the application name is placed in the title bar.

DefaultResponse - Optional. **String** expression displayed in the text box as the default response if no other input is provided. If you omit *DefaultResponse*, the displayed text box is empty.

XPos - Optional. Numeric expression that specifies, in twips, the distance of the left edge of the dialog box from the left edge of the screen. If you omit *XPos*, the dialog box is centered horizontally.

YPos- Optional. Numeric expression that specifies, in twips, the distance of the upper edge of the dialog box from the top of the screen. If you omit *YPos*, the dialog box is positioned vertically approximately one-third of the way down the screen.

To specify more than the first argument, you must use the **InputBox** function in an expression. If you omit any positional arguments, you must retain the corresponding comma delimiter.

Note The **InputBox** function requires UIPermission at the SafeTopLevelWindows level, which may affect its execution in partial trust situations.

Example: This example shows various ways to use the **InputBox** function to prompt the user to enter a value. If the x and y positions are omitted, the dialog box is automatically centered for the respective axes. The variable MyValue contains the value entered by the user if the user clicks OK or presses the ENTER key. If the user clicks Cancel, a zero-length string is returned.

```
Dim message, title, defaultValue As String
Dim myValue As Object
message = "Enter a value between 1 and 3"    ' Set prompt.
title = "InputBox Demo"    ' Set title.
DefaultValue = "1"    ' Set default value.

' Display message, title, and default value.
myValue = InputBox(message, title, defaultValue)
```

Notes

```
' Display dialog box at position 100, 100.
myValue = InputBox(message, title, defaultValue, 100, 100)
```

11.6 Dialog Result Class

It defines constants that represent common results returned from dialog boxes.

```
Enum
  |
  +—DialogResult
Syntax:
public class DialogResult
extends Enum
```

If you assign a DialogResult value to the dialogResult property of a Button control, clicking the button closes the parent form and sets the form's dialogResult property to that value (if the form was displayed using the showDialog method).

11.6.1 Dialog Result Values

The DialogResult enum specifies identifiers to indicate the return value of a dialog box. Here are the possible values for DialogResult:

Table 11.3: DialogResult Values

Member Name	Description
None	Nothing is returned from the dialog box. This means that the modal dialog continuous running.
OK	The dialog box return value is OK (usually sent from a button labeled OK).
Cancel	This dialog box return value is Cancel (usually sent from a button labeled cancel).
Abort	This dialog box return value is Abort (usually sent from a button labeled Abort).
Retry	This dialog box return value is Retry (usually sent from a button labeled Retry).
Ignore	The dialog box return value is Ignore (usually sent from a button labeled Ignore).
Yes	The dialog box return value is Yes (usually sent from a button labeled Yes).
No	The dialog box return value is No (usually sent from a button labeled No).

Source: http://msdn.microsoft.com/en-us/library/system.windows.forms.dialogresult.aspx

Case Study

Migration from VB6 to .NET

Abstract

The client is a US-based consulting company, holding a group of experienced developers who have worked for decades of years.

The client asked us to bring enhancements and modifications to the former VB6 application using .NET for their Securities client.

Situation

- The client possessed a legacy application developed with VB6. But it was hard for them to find resources within their organization specializing in VB6, thus it would

Contd...

Notes

not be cost-effective if the maintenance was carried out through the former architecture.

- As the former specification was missing, it is not very easy to understand the requirements just by analyzing the code.
- We were required to not only upgrade the code, but also transfer the database from Access being used to Microsoft SQL Server 2005. The legatary database was poorly designed without essential constraints, leading to many tables containing similar but inconsistent data in the database.

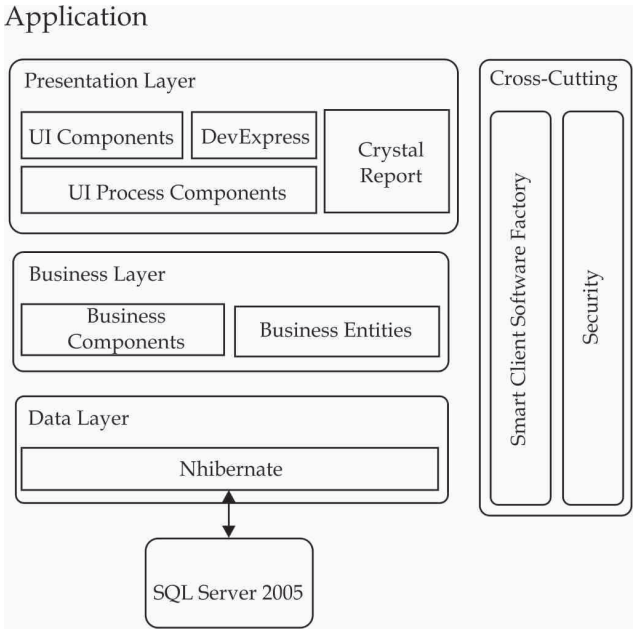
Solution

Technology required:

- .NET Framework 2.0
- Smart Client Software Factory
- NHibernate 1.2
- DevExpress 7.2
- Crystal Report

Because the client would like to rewrite the application with .NET solution, considering the scalability and a more reasonable architecture of the system, we adopted Smart Client Software Factory to create the framework of the whole system. Meanwhile, NHibernate was adopted as Data Access Layer, which not only speeded up the development progress, but also made the code clearer and more legible. In addition, DevExpress was utilized as the main controls of presentation tier which gave the application a modern, professional and easy-to-use look.

Figure 1



- Sufficient and effective communication was essential for this project as the specification was missing. In order to ensure our work right, periodical products were submitted to the client for confirmation, such as Model Design, Prototype and

Contd...

Notes

etc. We also discussed the specific and complex issues with the clients via Skype meeting frequently.

- We redesigned the database in SQL Server 2005 based on the business requirement. We merged all the similar but inconsistent data into one to ensure the data unique before importing data to our new database.

Benefits to the Client

Client has the following benefits:

- The new version of application can now meet all the requirements of the client, including the new features.
- The new system is established with excellent extensibility and maintainability. The client is now able to add new functions by himself with ease in future.
- The new application is easier to use, and has fantastic face.
- The client updated the system in scheduled time.

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://www.novasoftware.com/Case_Studies/Cases/Migration_from_VB6_to_dot_NET.aspx

Self Assessment

Fill in the blanks:

11. ..... Class displays a message box that can contain text, buttons, and symbols that inform and instruct the user.
12. A ..... is a special dialog box used to display a piece of information to the user.
13. The return value of the MsgBox() function is based on the ..... enumeration.
14. ..... defines constants that represent common results returned from dialog boxes.
15. The DialogResult ..... specifies identifiers to indicate the return value of a dialog box.

11.7 Summary

- A modal dialog box is displayed by a function when the function needs additional data from a user to continue.
- A modeless dialog box, on the other hand, does not prevent a user from activating other windows while it is open.
- The dialog box control classes inherit from the CommonDialog class and override the RunDialog() function of the base class.
- The RunDialog() function is automatically invoked when a user of a dialog box calls its ShowDialog() function.
- The ShowDialog method is used to display all the dialog box controls at run time.

Notes

- The Common Dialog Box Library provides a creation function and a structure for each type of common dialog box.
- The OpenFileDialog control prompts the user to open a file and allows the user to select a file to open.
- The OpenFileDialog control class inherits from the abstract class FileDialog.
- A characteristic of dialog boxes is that they provide an OK and a Cancel button.
- The SaveFileDialog control prompts the user to select a location for saving a file and allows the user to specify the name of the file to save data.
- The Color Dialog Box displays a modal dialog box that allows the user to choose a specific color.
- Message Box Class displays a message box that can contain text, buttons, and symbols that inform and instruct the user.
- Dialog Result Class defines constants that represent common results returned from dialog boxes.

11.8 Keywords

Color Dialog Box: It displays a modal dialog box that allows the user to choose a specific color.

Common Dialog Box Library: It provides a creation function and a structure for each type of common dialog box.

Dialog Result Class: It defines constants that represent common results returned from dialog boxes.

Message Box Class: It displays a message box that can contain text, buttons, and symbols that inform and instruct the user.

Modal dialog box: It is displayed by a function when the function needs additional data from a user to continue.

Modeless dialog box: It does not prevent a user from activating other windows while it is open.

OpenFileDialog control: It prompts the user to open a file and allows the user to select a file to open.

RunDialog() function: It is automatically invoked when a user of a dialog box calls its ShowDialog() function.

SaveFileDialog control: It prompts the user to select a location for saving a file and allows the user to specify the name of the file to save data.

ShowDialog method: It is used to display all the dialog box controls at run time.

11.9 Review Questions

1. What are Common Dialog Boxes?
2. Differentiate between modal and modeless dialog boxes.
3. How to use a Common Dialog Box?
4. Write a note on Common Dialog Box Library.
5. Explain the Common Dialog Control with example.

Notes

6. Why do we use OpenFileDialog?
7. Explain the SaveFileDialog.
8. Why do we use the ColorDialog Box?
9. Write a small note on MessageBox Class.
10. Explain the use of Dialog Result Class with a suitable example.

Answers: Self Assessment

1.	Modal	2.	RunDialog
3.	ShowDialog	4.	CommonDialogBox
5.	True	6.	False
7.	SaveFileDialog	8.	FileDialog
9.	False	10.	True
11.	MessageBox	12.	MessageBox()
13.	MsgBoxResult	14.	DialogResult Class
15.	Enum		

11.10 Further Readings

Books

Building Distributed Applications with Visual Basic.NET, Dan Fox, Sams.

Object-oriented Programming with Visual Basic.Net, Hamilton.

Programming Visual Basic .NET, J. Liberty.

VB .NET Language in a Nutshell, Steven Roman, Ron Petrusha, Paul Lomax.

Online links

http://www.exforsys.com/tutorials/vb.net-2005/dialog-boxes-in-visual-basic-.net-2005.html

http://msdn.microsoft.com/en-us/library/aa969773.aspx

http://vbadud.blogspot.in/2007/06/visual-basic-common-dialog.html

http://www.functionx.com/vbnet/controls/colordialog.htm

http://www.homeandlearn.co.uk/net/nets4p6.html

Notes

Unit 12: File Input/Output

CONTENTS

Objectives

After studying this unit, you will be able to:

- Understand working with files
- Discuss the working with directory
- Explain the system.IO operations

Introduction

When you first start learning VB.NET, one of the first things you may notice is the absence of "traditional" file I/O support in .NET. Microsoft has replaced the classic IO operations by stream operations. A stream is a simple concept that originated in the Unix world. You can think of

Notes

stream as a channel through which data flows from your application to a sequential data store (such as a file, a string, a byte array, or another stream), or vice versa. To understand why the traditional file I/O operations were replaced by streams, you must consider that not all data reside in files. Modern applications acquire data from many different data stores, including files, in-memory buffers and the Internet. The stream analogy enables applications to access all these data stores with the same programming model. There's no need to learn how to use Sockets to access a file on a remote Web server. You can establish a stream between your application and a remote resource and read the bytes as the server sends them. A stream encapsulates all the operations you can perform against a data store. The big advantage is that after you learn how to deal with streams for one data source, you can apply the same techniques to widely differing data sources.

The Stream class is abstract; you can't declare a new instance of type Stream in your code. There are five classes in the .NET Framework that derive from the Stream class. These are:

- ***FileStream*:** Supports sequential and random access to files.
- ***MemoryStream*:** Supports sequential and random access to memory buffers.
- ***NetworkStream:*** Supports sequential access to Internet resources. The NetworkStream resides in the System.Net.Sockets namespace.
- ***CryptoStream:*** Supports data encryption and decryption. The CryptoStream resides in the System.Security.Cryptography namespace.
- ***BufferedStream*:** Supports buffered access to stream that do not support buffering on their own.

Not all streams support exactly the same operations. A stream for reading a local file, for example, can report the length of the file and the current position in the file, with the *Length* and *Position* properties, respectively. You can jump to any location in the file with the *Seek* method. In contrast, a stream for reading a remote file doesn't support those features. But the stream classes help you differentiate Streams programmatically, by providing *CanSeek*, *CanRead* and *CanWrite* properties. Despite some data-store-dependent differences, the basic methods of all Stream classes let you write data to or read data from the underlying data store.

12.1 Working with Files

A file is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a stream. The stream is basically the sequence of bytes passing through the communication path. There are two main streams: the input stream and the output stream. The input stream is used for reading data from file (read operation) and the output stream is used for writing into the file (write operation).

12.1.1 File Class

It provides static methods for the creation, copying, deletion, moving, and opening of files, and aids in the creation of FileStream objects.

```
Syntax:
<ComVisibleAttribute(True)> _
Public NotInheritable Class File
```

Use the File class for typical operations such as copying, moving, renaming, creating, opening, deleting, and appending to files. You can also use the File class to get and set file attributes or DateTime information related to the creation, access, and writing of a file. Many of the File methods return other I/O types when you create or open files. You can use these other types to

Notes

further manipulate a file. Because all File methods are static, it might be more efficient to use a File method rather than a corresponding FileInfo instance method if you want to perform only one action. All File methods require the path to the file that you are manipulating. The static methods of the File class perform security checks on all methods. If you are going to reuse an object several times, consider using the corresponding instance method of FileInfo instead, because the security check will not always be necessary. By default, full read/write access to new files is granted to all users.

Table 12.1 describes the enumerations that are used to customize the behavior of various File methods.

Table 12.1: Enum Used in File Method

Enumeration	Description
FileAccess	Specifies read and write access to a file.
FileShare	Specifies the level of access permitted for a file that is already in use.
FileMode	Specifies whether the contents of an existing file are preserved or overwritten, and whether requests to create an existing file cause an exception.

Source: http://msdn.microsoft.com/en-us/library/system.io.file.aspx

12.1.2 Path

In members that accept a path as an input string, that path must be well-formed or an exception is raised. For example, if a path is fully qualified but begins with a space, the path is not trimmed in methods of the class. Therefore, the path is malformed and an exception is raised. Similarly, a path or a combination of paths cannot be fully qualified twice. For example, "c:\temp c:\windows" also raises an exception in most cases. Ensure that your paths are well-formed when using methods that accept a path string. In members that accept a path, the path can refer to a file or just a directory. The specified path can also refer to a relative path or a Universal Naming Convention (UNC) path for a server and share name.

Example: All the following are acceptable paths:

- "c:\\MyDir\\MyFile.txt" in C#, or "c:\MyDir\MyFile.txt" in Visual Basic.
- "c:\\MyDir" in C#, or "c:\MyDir" in Visual Basic.
- "MyDir\\MySubdir" in C#, or "MyDir\MySubDir" in Visual Basic.
- "\\\\MyServer\\MyShare" in C#, or "\\MyServer\MyShare" in Visual Basic.

VB.NET provides effective ways of dealing with filenames and paths by using System.IO namespace . The Path Class performs operations on String instances that contain file or directory path information. Path Class returns a string that can contain absolute or relative location information.

The following are the some important operations in VB.Net Path Class:

- ***GetDirectoryName*** **-** Returns the directory information for the specified path string.
- ***GetExtension*** **-** Returns the extension of the specified path string.
- ***GetFileName*** **-** Returns the file name and extension of the specified path string.
- ***GetFileNameWithoutExtension*** **-** Returns the file name of the specified path string without the extension.
- ***GetFullPath*** **-** Returns the absolute path for the specified path string.

Notes

Example:

```
Imports System.IO

Public Class Form1

   Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
      Dim tmpPath As String
      Dim rootPath As String
      Dim filename As String
      Dim extension As String
      Dim directory As String
      Dim fullPath As String
      Dim filenameWithoutExtension As String

      tmpPath = "c:\\windows\\inf\\wvmic.inf"

      rootPath = Path.GetPathRoot(tmpPath)
      filename = Path.GetFileName(tmpPath)
      extension = Path.GetExtension(tmpPath)
      directory = Path.GetDirectoryName(tmpPath)
      filenameWithoutExtension = Path.GetFileNameWithoutExtension(tmpPath)
      fullPath = Path.GetFullPath(tmpPath)
      MsgBox(directory)
   End Sub
End Class
```

12.1.3 File Enumerations

An Enum type stores special values. These are named constants. With an Enum type we can remove magic constants throughout our program. And this improves code clarity. It makes the program easier to maintain. We will be discussing certain enums that deal with files in VB.NET.

FileAccess

In order to perform an operation on a file, you must specify to the operating system how to proceed. One of the options you have is to indicate the type of access that will be granted on the file. This access is specified using the FileAccess enumerator. The members of the FileAccess enumerator are:

- ***FileAccess.Write:*** New data can be written to the file
- ***FileAccess.Read:*** Existing data can be read from the file
- ***FileAccess.ReadWrite:*** Existing data can be read from the file and new data be written to the file

FileAttributes

It provides attributes for files and directories.This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

Notes

```
Syntax:
<SerializableAttribute> _
<FlagsAttribute> _
<ComVisibleAttribute(True)> _
Public Enumeration FileAttributes
```

Some members of the FileAttributes enum are:

Table 12.2: Members of the FileAttributes Enum

Member name	Description
ReadOnly	The file is read-only.
Hidden	The file is hidden, and thus is not included in an ordinary directory listing.
System	The file is a system file. That is, the file is part of the operating system or is used exclusively by the operating system.
Directory	The file is a directory.
Archive	The file is a candidate for backup or removal.
Device	Reserved for future use.
Normal	The file is a standard file that has no special attributes. The attribute is valid only if it is used alone.
Temporary	The file is a temporary. A temporary file contains data is needed while an application is executing but is not needed after the application is finished. File systems try to keep all the data in memory for quicker access rather than flushing the data back to ass storage. A temporary file should be deleted by the application as soon as it is no longer needed.
SparseFile	The file is sparse file. Sparse files are typically larger files whose data consists of mostly zeros.
ReparsePoint	The file contains a reparse point, which is a block of user-defined data associated with a file or a directory.

Source: http://msdn.microsoft.com/en-us/library/system.io.fileattributes.aspx

Example:

The following example shows how to retrieve the attributes for a file and check if the file is read-only.

```
Imports System.IO
Imports System.Text

Module Module1
     Sub Main()
            Dim attributes = File.GetAttributes("c:/Temp/testfile.txt")
                  If ((attributes And FileAttributes.ReadOnly) =
FileAttributes.ReadOnly) Then
                Console.WriteLine("read-only file")
         Else
                Console.WriteLine("not read-only file")
         End If
     End Sub
End Module
FileMode
It specifies how the operating system should open a file.
```

Notes

```
Syntax:

<SerializableAttribute> _
<ComVisibleAttribute(True)> _
Public Enumeration FileMode
```

Some of the members are:

Table 12.3: Members of the FileMode Enum

Member name	Description
CreateNew	Specifies that the operating system should create a new file. This requires FileIOPermissionAccess.Write permission. If the file already exists, and IOException exception is thrown.
Create	Specifies that the operating system should create a new file. If the file already exists, it will be overwritten. This requires FileIOPermissionAccess.Write permission. FileMode.Create is equivalent to requesting that if the file does not exist, use CreateNew; otherwise, use Truncate. If the file already exists but is a hidden file, an UnauthorizedAccessException exception is thrown.
Open	Specifies that the operating system should open an existing file. The ability to open the file is dependent on the value specified by the FileAccess enumeration. A System.IO.FileNotFoundException exception is thrown if the file does not exist.
OpenOrCreate	Specifies that the operating system should open a file if it exists; otherwise, a new file should be created. If the file is opened with FileAccess.Read, FileIOPermissionAccess.Read permission is required. If the file access is FileAccess.Write, FileIOPermissionAccess.Write permission is required. If the file is opened with FileAccess.ReadWrite, both FileIOPermissionAccess.Read and FileIOPermissionAccess.Write permission are required.
Truncate	Specifies that the operating system should open an existing file. When the file is opened, it should be truncated so that its size is zero bytes. This requires FileIOPermissionAccess.Write permission. Attempts to read from a file opened with FileMode.Truncate cause an ArguentException exception.
Append	Opens the file if it exists and seeks to the end of the file, or creates a new file. This requires FileIOPermissionAccess.Append permission. FileMode.Append can be used only in conjunction file with FileAccess.Write. Trying to seek a position before the end of the file throws an IOException exception, and any attempt to read fails and throws a NotSupportException exception.

Source: http://msdn.microsoft.com/en-us/library/system.io.filemode.aspx

A FileMode parameter is specified in many of the constructors for FileStream, IsolatedStorageFileStream, and in the Open methods of File and FileInfo to control how a file is opened. FileMode parameters control whether a file is overwritten, created, opened, or some combination thereof. Use Open to open an existing file. To append to a file, use Append. To truncate a file or create a file if it doesn't exist, use Create.

Example: The following FileStream constructor opens an existing file (FileMode.Open).

```
Dim s2 As New FileStream(name, FileMode.Open, FileAccess.Read, FileShare.Read)
```

FileShare

FileShare enumerators have the following members:

- ***Inheritable:*** It allows a file handle to pass inheritance to the child processes
- ***None:*** It declines sharing of the current file

Notes

- ***Read:*** It allows opening the file for reading
- ***ReadWrite:*** It allows opening the file for reading and writing
- ***Write:*** It allows opening the file for writing

A typical use of this enumeration is to define whether two processes can simultaneously read from the same file. For example, if a file is opened and Read is specified, other users can open the file for reading but not for writing. A FileShare parameter is specified in some of the constructors for FileStream, IsolatedStorageFileStream, and in some of the Open methods of File and FileInfo to control how a file is opened.

Example: The following FileStream constructor opens an existing file and grants read-only access to other users (Read).

```
Dim s2 As New FileStream(name, FileMode.Open, FileAccess.Read, FileShare.Read)
```

12.1.4 Basic File Class Operations

Some of the operations possible with files in VB.NET are discussed below.

How to Create a File

The File class can be used to create a new file. To support this operation, the File class is equipped with the Create() method that is overloaded with two versions as follows:

```
Public Shared Function Create(path As String) As FileStream
Public Shared Function Create(path As String, bufferSize As Integer) As
FileStream
```

In both cases, the File.Create() method returns a Stream value, which is a FileStream value. As the File.Create() method indicates, it takes the name or path of the file as argument. If you know or want to specify the size, in bytes, of the file, you can use the second version.

To provide the same operation of creating a file, you can use the Open() method of the File class. It is overloaded in three versions as follows:

```
Public Shared Function Open ( _
    path As String, _
    mode As FileMode _
) As FileStream

Public Shared Function Open ( _
    path As String, _
    mode As FileMode, _
    access As FileAccess _
) As FileStream
Public Shared Function Open ( _
    path As String, _
    mode As FileMode, _
    access As FileAccess, _
    share As FileShare _
) As FileStream
```

Notes

How to Copy a File

You can also copy a file that you've created. This time, we don't need the StreamWriter or StreamReader of System.IO. We need the File object:

```
System.IO.File
```

This just means "System.IO has an object called File. Use this File object".

File has it's own properties and methods you can use. One of these is Copy. Here's some code that makes a copy of our test file.

```
Dim FileToCopy As String
Dim NewCopy As String
FileToCopy = "C:\Users\Owner\Documents\test.txt"
NewCopy = "C:\Users\Owner\Documents\NewTest.txt"
If System.IO.File.Exists( FileToCopy ) = True Then
System.IO.File.Copy( FileToCopy, NewCopy )
MsgBox("File Copied")
End If
```

The file we want to copy is called "test.txt". We've put this inside of a string variable called FileToCopy. The name of the new file we want to create, and its location, are assigned to a variable called NewCopy.

Next, we have to check to see if the file we're trying to copy exists. Only if it does should we go ahead and copy it. You've met this code before. Inside of the If Statement, we have this:

```
System.IO.File.Copy( FileToCopy, NewCopy )
```

We use the Copy method of System.IO.File. In between the round brackets, you first type the name of the file you want to copy. After a comma, you then type the name of the new file and its new location.

How to Delete a File

Deleting a file is quite simple – but dangerous! So be very careful when you're trying out this code. Make sure the file you're going to delete is not needed – you won't be able to restore it from the recycle bin!

To delete a file from your computer, you use the Delete method of System.IO.

Example:

```
Dim FileToDelete As String
FileToDelete = "C:\Users\Owner\Documents\testDelete.txt"
If System.IO.File.Exists( FileToDelete ) = True Then
System.IO.File.Delete( FileToDelete )
MsgBox("File Deleted")
End If
```

First, we've set up a string variable called FileToDelete. We've then assigned the name of a file to this variable - "C:\testDelete.txt". (We created this file first, and made sure that it was safe to junk it!)

Next, we test to see if the File Exists. In the IF Statement, we then had this:

```
System.IO.File.Delete( FileToDelete )
```

Notes

After selecting the Delete method, you type the name of the file you want to get rid of. This goes between a pair of round brackets.

Getting and Setting File Attributes and Access Information on Files

The System.IO namespaces contain types that support input and output, including the ability to read and write data to streams either synchronously or asynchronously.

FileInfo Class Provides instance methods for the creation, copying, deletion, moving, and opening of files. FileSystemInfo.Attributes property is used to gets or sets the attributes for the current file or directory.

```
_file.Attributes = IO.FileAttributes.ReadOnly
```

Example: The following vb.net program shows how to set read-only and hidden property to a file.

```
Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Try
            Dim _file As IO.FileInfo =
My.Computer.FileSystem.GetFileInfo("c:\test.txt")
            _file.Attributes = IO.FileAttributes.ReadOnly
            _file.Attributes = IO.FileAttributes.Hidden
        Catch ex As System.IO.FileNotFoundException
            MsgBox(ex.ToString())
        End Try
    End Sub

End Class
```

Moving Files Around

You move a file in a similar manner as you did to Copying a File – specify a source file and a new destination for it. This time, we use the Move method of System.IO.File.

Example:

```
Dim FileToMove As String
Dim MoveLocation As String
FileToMove = "C:\Users\Owner\Documents\test.txt"
MoveLocation = "C:\Users\Owner\Documents\TestFolder\test.txt"
If System.IO.File.Exists( FileToMove ) = True Then
System.IO.File.Move( FileToMove, MoveLocation )
MsgBox("File Moved")
End If
```

The above code assumes that you have created a folder on your hard drive called "TestFolder":

```
MoveLocation = "C:\Users\Owner\Documents\TestFolder\test.txt"
```

The file called test.txt will then be moved inside of this new location. You can give it a new name, if you want. In which case, just change the name of the file when you're moving it:

```
MoveLocation ="C:\Users\Owner\Documents\TestFolder\NewName.txt"
```

Notes

Again though, the thing to type in the round brackets of the method is first the Source file, then the Destination.

```
System.IO.File.Move( FileToMove, MoveLocation )
```

Self Assessment

Fill in the blanks:

1. The Stream class is ..... type of class.
2. ..... class supports sequential and random access to files.
3. A ..... is a collection of data stored in a disk with a specific name and a directory pat.
4. The ..... is basically the sequence of bytes passing through the communication path.
5. ..... Class provides static methods for the creation, copying, deletion, moving, and opening of files, and aids in the creation of FileStream objects.
6. The ..... Class performs operations on String instances that contain file or directory path information.

12.2 Working with Directory

Using Directory class, we can create, delete, move, etc. operations in VB.NET. Because of the static nature of Directory class, we do not have to instantiate the class. We can call the methods in the class directly from the Directory class.

Table 12.4: Static Methods of Directory Class

Directory Class (static methods)	DirectoryInfo Class	Use
CreateDirectory	Create	Create a directory.
Delete	Delete	Delete a directory.
Exists	Exists (property)	Check whether a directory exists. ***Note:*** The Directory class implements a method Exists while the DirectoryInfo class implements Exists as a property.
Move	MoveTo	Move a directory to the destination path.
GetDirectories	GetDirectories	Get subdirectories within a directory.
GetLogicalDrives	Root (property)	Get the root of a directory. ***Note:*** DirectoryInfo class implements Root as a property.
GetParent	Parent (property)	Get the parent directory. ***Note:*** DirectoryInfo class implements Parent as a property.

Source: http://www.dotnetheaven.com/article/directory-and-directoryinfo-classes-in-vb.net

Like the FileInfo class, the DirectoryInfo class also has a single constructor that takes either the full path or relative path to the directory as the input parameter:

```
Dim d1 As New DirectoryInfo("c:\temp") //Make a directory object
```

Notes

Some important properties of the DirectoryInfo class, such as CreationTime, Exists, FullName, LastAccessTime, LastWriteTime, Name, Parent, and Root, work as their names suggest. For example, CreationTime shows the creation time of the directory. FullName is the full qualified path of the directory, while Name is just the relative folder name without the path (e.g., a directory with the FullName of c:\My Project\test would have a Name of test).

FileInfo Class

The FileInfo class is derived from the FileSystemInfo class. It has properties and instance methods for creating, copying, deleting, moving, and opening of files, and helps in the creation of FileStream objects. This class cannot be inherited. Following are some commonly used properties of the FileInfo class:

- ***Attributes:*** Gets the attributes for the current file.
- ***CreationTime:*** Gets the creation time of the current file.
- ***Directory:*** Gets an instance of the directory which the file belongs to.
- ***Exists:*** Gets a Boolean value indicating whether the file exists.
- ***Extension:*** Gets the string representing the file extension.
- ***FullName:*** Gets the full path of the file.
- ***LastAccessTime:*** Gets the time the current file was last accessed.
- ***LastWriteTime:*** Gets the time of the last written activity of the file.
- ***Length:*** Gets the size, in bytes, of the current file.
- ***Name:*** Gets the name of the file.

Following are some commonly used methods of the FileInfo class:

- ***Public Function AppendText As StreamWriter:*** Creates a StreamWriter that appends text to the file represented by this instance of the FileInfo.
- ***Public Function Create As FileStream:*** Creates a file.
- ***Public Overrides Sub Delete:*** Deletes a file permanently.
- ***Public Sub MoveTo (destFileName As String):*** Moves a specified file to a new location, providing the option to specify a new file name.
- ***Public Function Open (mode As FileMode) As FileStream:*** Opens a file in the specified mode.
- ***Public Function Open (mode As FileMode, access As FileAccess) As FileStream:*** Opens a file in the specified mode with read, write, or read/write access.
- ***Public Function Open (mode As FileMode, access As FileAccess, share As FileShare) As FileStream:*** Opens a file in the specified mode with read, write, or read/write access and the specified sharing option.
- ***Public Function OpenRead As FileStream:*** Creates a read-only FileStream.
- ***Public Function OpenWrite As FileStream:*** Creates a write-only FileStream.

DirectoryInfo

The DirectoryInfo class is derived from the FileSystemInfo class. It has various methods for creating, moving, and browsing through directories and subdirectories. This class cannot be inherited.

Notes

Following are some commonly used properties of the DirectoryInfo class:

- ***Attributes:*** Gets the attributes for the current file or directory.
- ***CreationTime:*** Gets the creation time of the current file or directory.
- ***Exists:*** Gets a Boolean value indicating whether the directory exists.
- ***Extension:*** Gets the string representing the file extension.
- ***FullName:*** Gets the full path of the directory or file.
- ***LastAccessTime:*** Gets the time the current file or directory was last accessed.
- ***Name:*** Gets the name of this DirectoryInfo instance.

Following are some commonly used methods of the DirectoryInfo class:

- ***Public Sub Create:*** Creates a directory.
- ***Public Function CreateSubdirectory (path As String) As DirectoryInfo:*** Creates a subdirectory or subdirectories on the specified path. The specified path can be relative to this instance of the DirectoryInfo class.
- ***Public Overrides Sub Delete:*** Deletes this DirectoryInfo if it is empty.
- ***Public Function GetDirectories As DirectoryInfo() :*** Returns the subdirectories of the current directory.
- ***Public Function GetFiles As FileInfo():*** Returns a file list from the current directory.

12.2.1 Using the Classic File System Object

The File System Object (FSO) enables you to manipulate the files, folders and drives as well as read and write to sequential files. Before using the FSO, you have to add the "Microsoft Scripting Runtime Library" to the current project by selecting "Project", "References" from the menu bar. Alternatively you can use the CreateObject function *to* create the reference at run-time.

There are five types of File System Object:

1. File
2. Folder
3. Drive
4. TextStream
5. Random Access Files

The FileSystemObject is used to manipulate the files, folders and directories. The following is a list of some of the methods available to the FileSystemObject.

File System Object Methods:

Table 12.5: FSO Methods

Method	Description
CopyFile	Used to copy an existing file.
CopyFolder	Used to copy an existing folder.
CreateFolder	Used to create a folder.

Contd...

Notes

CreateTextFile	Used to create a text file.
DeleteFile	Used to delete a file.
DeleteFolder	Used to delete a folder
DriveExists	Used to determine whether a drive exists.
FileExists	Used to determine whether a file exists.
FolderExists	Used to determine whether a folder exists.
GetAbsolutePathName	Used to return the full path name.
GetDrive	Used to return a specified drive.
GetDriveName	Used to return a specified drive name.
GetFile	Used to return a specified file.
GetFileName	Used to return the file name.
GetFolder	Used to return a specified folder.
GetParentFolderName	Used to return the name of the parent folder.
GetTempName	Used to create and return a string representing a file name.
MoveFolder	Used to move a folder.
OpenTextFile	Used to open an existing text file.

Source: http://www.virtualsplat.com/tips/visual-basic-fso.asp

FSO Folder Object

The following uses some of the FSO Folder's properties to display information about a folder.

Example:

```
Private Sub displayFolderInfo(ByVal folderName As String)
       Dim fso As New FileSystemObject
       Dim folderSpec As Folder
       Dim strInfo As String
        Set folderSpec = fso.GetFolder(folderName)
        strInfo = folderSpec.Name & vbCrLf
       strInfo = strInfo & "Created: "
        strInfo = strInfo & folderSpec.DateCreated & vbCrLf
       strInfo = strInfo & "Size: "
        strInfo = strInfo & folderSpec.Size
        MsgBox strInfo, vbInformation, "Folder Information"
       Set folderSpec = Nothing
End Sub
```

The Copy, CreateTextFile, Delete, and Move methods are available for the FSO Folder object.

FSO Drive Object

The following example iterates through the Drives collection and writes the drive name for each drive found.

Example:

```
Dim fso As New FileSystemObject
Dim connectedDrives As drives, drv As Drive
Dim strInfo As String, driveName As String
```

Notes

```
Set connectedDrives = fso.drives
On Error Resume Next
For Each drv In connectedDrives
       strInfo = strInfo & drv.DriveLetter & ": "
       ' Check if the drive is shared
       If drv.DriveType = 3 Then
             driveName = drv.ShareName
      Else
             driveName = drv.VolumeName
      End If
       strInfo = strInfo & driveName
       strInfo = strInfo & " Free space: " & drv.FreeSpace
       If drv.IsReady Then
             strInfo = strInfo & " ready" & vbCrLf
      Else
             strInfo = strInfo & " not ready" & vbCrLf
      End If
      Next drv
MsgBox strInfo, vbInformation, "Connected Drives"
Set connectedDrives = Nothing
Set fso = Nothing
```

FSO TextStream Object

The FSO TextStream object is used to read and write to sequential text files.

Random Access Files: VB.NET uses fixed length records in order to implement random access files. Data may be inserted into the file without destroying any other data in the file. Data may also be amended or deleted without having to rewrite the entire file, which is the case with sequential files.

12.2.2 Notify Filters

It specifies changes to watch for in a file or folder. This enumeration has a FlagsAttribute attribute that allows a bitwise combination of its member values.

```
Syntax:
<FlagsAttribute> _
Public Enumeration NotifyFilters
```

Members:

- ***FileName:*** The name of the file.
- ***DirectoryName:*** The name of the directory.
- ***Attributes:*** The attributes of the file or folder.
- ***Size:*** The size of the file or folder.
- ***LastWrite:*** The date the file or folder last had anything written to it.
- ***LastAccess:*** The date the file or folder was last opened.
- ***CreationTime:*** The time the file or folder was created.
- ***Security:*** The security settings of the file or folder.

Notes

You can combine the members of this enumeration to watch for more than one kind of change. For example, you can watch for changes in the size of a file or folder, and for changes in security settings. This raises an event anytime there is a change in size or security settings of a file or folder.

Example: The following example creates a FileSystemWatcher to watch the directory that is specified at runtime. The component is set to watch for any changes in LastWrite and LastAccess time, the creation, deletion, or renaming of text files in the directory. If a file is changed, created, or deleted, the path to the file prints to the console. When a file is renamed, the old and new paths print to the console.

```
Imports System
Imports System.IO
Imports Microsoft.VisualBasic
Imports System.Security.Permissions

Public Class Watcher

      Public Shared Sub Main()

          Run()

     End Sub

      <PermissionSet(SecurityAction.Demand, Name:="FullTrust")> _
      Private Shared Sub Run

         Dim args() As String = System.Environment.GetCommandLineArgs()
           ' If a directory is not specified, exit the program.
          If args.Length <> 2 Then
               ' Display the proper way to call the program.
                Console.WriteLine("Usage: Watcher.exe (directory)")
             Return
         End If

           ' Create a new FileSystemWatcher and set its properties.
           Dim watcher As New FileSystemWatcher()
          watcher.Path = args(1)
           ' Watch for changes in LastAccess and LastWrite times, and
           ' the renaming of files or directories.
               watcher.NotifyFilter = (NotifyFilters.LastAccess Or
NotifyFilters.LastWrite      Or      NotifyFilters.FileName      Or
NotifyFilters.DirectoryName)
          ' Only watch text files.
          watcher.Filter = "*.txt"

          ' Add event handlers.
            AddHandler watcher.Changed, AddressOf OnChanged
            AddHandler watcher.Created, AddressOf OnChanged
            AddHandler watcher.Deleted, AddressOf OnChanged
            AddHandler watcher.Renamed, AddressOf OnRenamed
```

Notes

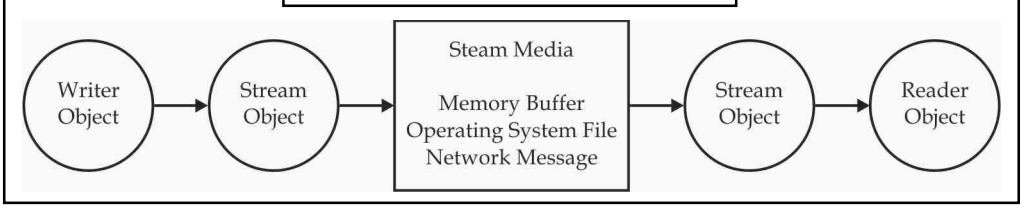
```
            ' Begin watching.
             watcher.EnableRaisingEvents = True

            ' Wait for the user to quit the program.
             Console.WriteLine("Press 'q' to quit the sample.")
            While Chr(Console.Read()) <> "q"c
           End While
     End Sub

      ' Define the event handlers.
      Private Shared Sub OnChanged(source As Object, e As FileSystemEventArgs)
           ' Specify what is done when a file is changed, created, or deleted.
             Console.WriteLine("File: " & e.FullPath & " " & e.ChangeType)
     End Sub

      Private Shared Sub OnRenamed(source As Object, e As RenamedEventArgs)
           ' Specify what is done when a file is renamed.
             Console.WriteLine("File: {0} renamed to {1}", e.OldFullPath,
e.FullPath)
     End Sub

End Class
```

12.2.3 Streams

Stream is a sequence of bytes, so the details of writing and reading data can be abstracted. For example, you can write a generic piece of code that writes an XML document to a stream. Then you can use that code in an application to write the XML document to several different types of streams, such as an in-memory buffer, a file, or even a network connection. While a stream object works at the byte level, it's fortunate that programmers aren't typically required to write and read data on a byte-by-byte basis. Imagine how hard it would be to take the data associated with a high-level document, such as an invoice, and transform it into a byte array. While that sounds like a challenging task, it would be even more difficult to write the code to reassemble the invoice from the byte array when it's time to present this data to the user.

The .NET Framework Class Library includes several complementary pairs of reader and writer objects that improve the usefulness and ease of use of streams. The Figure 12.1 illustrates the typical relationship between writer objects, stream objects, and reader objects.

Figure 12.1: Reader and Writer Objects



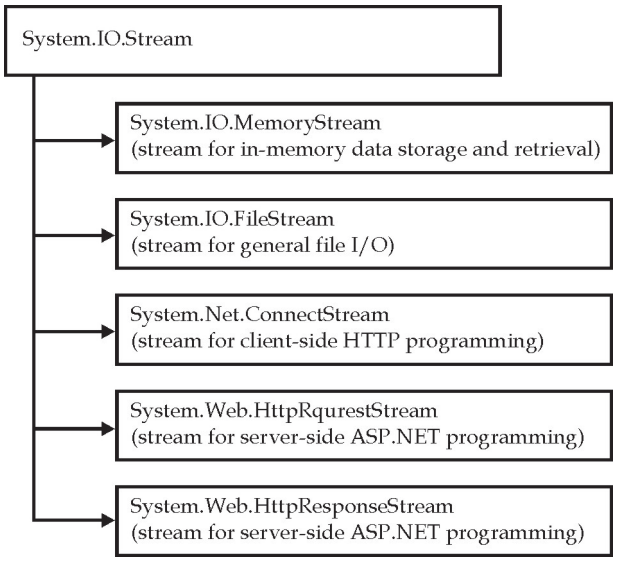
Source: http://msdn.microsoft.com/en-us/magazine/cc163710.aspx

To write data into a stream in an efficient manner, create a writer object and initialize it so it is associated with a target stream object. A writer object exposes high-level methods (such as WriteLine) that allow you to think in terms of high-level data structures (such as strings) instead of concerning yourself with the nitty-gritty details of actually writing bytes. Behind the scenes, the writer object does the work of crunching the data into a byte array and writing it into the target stream. When it's time to retrieve the data, you simply call high-level methods on the

Notes

reader object (for example, ReadLine). The reader object responds by fetching bytes from the stream and reassembling the data back into its original form. Before looking at any code, let's discuss where stream objects come from. The .NET Framework Class Library contains a Stream class in the System.IO namespace. The Stream class itself is an abstract class (a MustInherit class), meaning it cannot be directly instantiated to create an object. Its sole purpose is to serve as a base class for other Stream-derived classes. Actual stream objects must be instantiated from creatable classes that inherit either directly or indirectly from the Stream class. A key point is that the Stream class has been designed to provide a common implementation and a standard programming contract for every stream object. The .NET Framework provides several concrete implementations of the Stream class, a subset of which are shown in the Figure 12.2. As you can see, each of these Stream classes inherits from the Stream class and extends it to provide a unique implementation of a stream.

Figure 12.2: Inheriting from the Stream Class



Source: http://msdn.microsoft.com/en-us/magazine/cc163710.aspx

Example: You can use the MemoryStream class to create stream objects to store and retrieve data from an in-memory buffer. You can use the FileStream class to create stream objects to read or write data in a file on the local file system. You can also use stream objects created from one of the network stream classes to read or write the body of a network message.

```
'*** create memory stream object
Dim stream1 As Stream = New MemoryStream(4)

'*** write data to stream
stream1.WriteByte(2)
stream1.WriteByte(4)
stream1.WriteByte(6)
stream1.WriteByte(8)

'*** clear stream buffer (if it has one)
stream1.Flush()

'*** read data from stream
```

Notes

```
stream1.Position = 0
Dim result As Integer
result = stream1.ReadByte()
Do While result <> -1
   Console.WriteLine(result)
   result = stream1.ReadByte()
Loop

'*** close stream as soon as work is complete
stream1.Close()
```

12.2.4 FileStream

The System.IO namespace is contain file handling in visual basic with a class library that supports string, character and file manipulation, these classes perform creating, copying, moving, and deleting files operation's with the help of properties, methods and events. 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.

Here we discuss about the FileStream Class.

Nine overloaded constructors help you gain finer control over the file states. The constructor most commonly used, shown in below example, helps set the various access permissions and creation states on the file through the use of the FileMode, FileAccess, and FileShare enumerations.

FileStream Constructor

Dim fs As New FileStream("file.doc", FileMode.Create, FileAccess.Write)

FileMode Enumeration

The FileMode enumeration helps you the set the mode in which you want to open the file. You can use these modes to set your file up for appending or overwriting or initial creation, as detailed in the Table 12.6.

Table 12.6: Enumeration FileMode

Enumeration	Use
Append	Opens a file, If it exists, and seeks to the end of file; if file does not exist, creates a new file.
Create	Creates a new file, overwriting the previous file, if it exists
CreateNew	Creates a new file
Open	Opens an existing file
OpenOrCreate	Opens the file, if it exists, or creates a new one
Truncate	Opens an existing file and truncates its size to zero bytes.

Source: http://www.dotnetheaven.com/article/filestream-in-vb.net

Note The Append mode can only be used when FileAccess.Write permission is set.

Notes

FileAccess Enumeration

With the FileAccess enumerations described in Table 12.7, you can set the mode of access to a file. It's never good to authorize more access than needed-or less access than needed, for that matter. Choose Read when you intend to read from a file and Write when you write to a file. Remember, though, that if you specify Read access and later try to write to the file, an exception will be raised. The same applies when you specify Write access and try to read the file later.

Table 12.7: Enumeration FileAccess

Enumeration	Use
Read	Allows you to only read from the file
ReadWrite	Allows you to read and write to a file
Write	Allows you to only write to a file

Source: http://www.dotnetheaven.com/article/filestream-in-vb.net

FileShare Enumeration

The FileShare enumeration, detailed in Table 12.8, is very important if you wish to share your file with other processes. For example, suppose you have an XML file acting as a database file for an ASP.NET application. If you don't specify the FileAccess enumeration, only one user can read from the XML database file at a time; other concurrent users encounter an error when accessing the database because the FileShare.None enumeration is implemented by default.

Table 12.8: Enumeration FileShare

Enumeration	Use
None	Gains exclusive access to the file; no other process can access the file until it is closed the reopened
Read	Allows subsequent opening of the file but the read-only purpose
ReadWrite	Allows subsequent opening of file for reading and writing
Write	Allows subsequent opening of file for write-only purposes

Source: http://www.dotnetheaven.com/article/filestream-in-vb.net

The short example given below demonstrates how you can work with the FileStream class.

Drag a Button and a RichTextBox control onto the form. Use the following code:

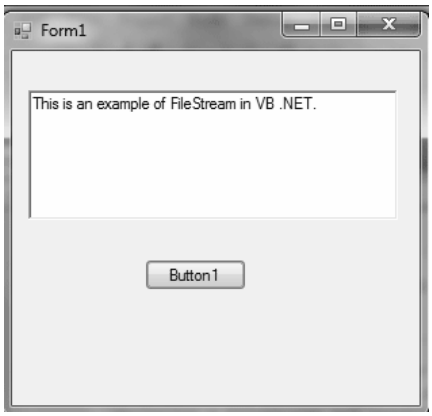
Example:

```
Imports System.IO
Imports System.Windows.Forms.Form
    Public Class Form1
           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 FileS As New FileStream("file.doc", FileMode.Create,
FileAccess.Write)
               'declaring a FileStream and creating a document file
```

Notes

```
named file with
                  'access mode of writing
                  Dim a As New StreamWriter(FileS)
                   'creating a new StreamWriter and passing the filestream
object fs as argument
                   a.WriteLine("This is an example of FileStream in VB
.NET.")
                  'writing text to the newly created file
                 a.Close()
                 'closing the file
                   FileS = New FileStream("file.doc", FileMode.Open,
FileAccess.Read)
                   'declaring a FileStream to open the file named file.doc
with access mode of reading
                  Dim i As New StreamReader(FileS)
                   'creating a new StreamReader and passing the filestream
object fs as argument
                   i.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 i.Peek() > -1
                       'peek method of StreamReader object tells how much
more data is left in the file
                      RichTextBox1.Text &= i.ReadLine()
                       'displaying text from doc file in the RichTextBox
                End While
                i.Close()
            End Sub
        End Class
Output:
```

Figure 12.3: Output of the Code Above



Source: http://www.dotnetheaven.com/article/filestream-in-vb.net

12.2.5 BufferedStream

The BufferedStream class also extends the Stream class. Buffers, or cached blocks of data in memory, provide speed and stability to the process of reading or writing because they prevent

Notes

numerous calls to the operating system. Buffered streams are used in conjunction with other streams to provide better read/write performance. The BufferedStream class can be used to either read data or write data but it cannot be used to perform both read and write operations together. The class has been optimized so that it maintains a suitable buffer at all times. When a buffer is not required, instead of slowing down the process, the class does not allocate any space in memory. File streams are already buffered and therefore a buffered stream is generally used to buffer network streams used in networking applications.

12.2.6 NetworkStream

The NetworkStream class provides methods for sending and receiving data over Stream sockets in blocking mode. You can use the NetworkStream class for both synchronous and asynchronous data transfer. To create a NetworkStream, you must provide a connected Socket. You can also specify what FileAccess permission the NetworkStream has over the provided Socket. By default, closing the NetworkStream does not close the provided Socket. If you want the NetworkStream to have permission to close the provided Socket, you must specify true for the value of the owns Socket parameter. Use the Write and Read methods for simple single thread synchronous blocking I/O. If you want to process your I/O using separate threads, consider using the BeginWrite and EndWrite methods, or the BeginRead and EndRead methods for communication.

The NetworkStream does not support random access to the network data stream. The value of the CanSeek property, which indicates whether the stream supports seeking, is always false; reading the Position property, reading the Length property, or calling the Seek method will throw a NotSupportedException.Read and write operations can be performed simultaneously on an instance of the NetworkStream class without the need for synchronization. As long as there is one unique thread for the write operations and one unique thread for the read operations, there will be no cross-interference between read and write threads and no synchronization is required.

12.2.7 CryptoStream

The common language runtime uses a stream-oriented design for cryptography. The core of this design is CryptoStream. Any cryptographic objects that implement CryptoStream can be chained together with any objects that implement Stream, so the streamed output from one object can be fed into the input of another object. The intermediate result (the output from the first object) does not need to be stored separately.

12.2.8 MemoryStream

A Memory Stream is created from an array of unsigned bytes rather than from a file or other stream. Memory streams are used as temporary, in-memory storage (temporary buffers) in lieu of creating temporary files. This stream is highly optimized for speed since the data is stored in memory and the processor can easily access it. Memory streams should be used to store frequently accessed data. The Read and Write methods of the MemoryStream class read and write from an internal buffer that is created when the memory stream is created.

12.2.9 Readers and Writers

To be able to open a file and read the data from a storage unit of a computer, such as a hard drive and able to save the data into the storage unit are important functions of a computer program. In fact, the ability to store, retrieve and modify data makes a computer a powerful tool in database management.

Notes

***Reading a Text File*:** Using text file is an easy way to manage data, although it is not as sophisticated as full fledged database management software such as SQL Server, Microsoft Access and Oracle. VB.NET allows the user to create a text file, save the text file as well as read the text file.

Reading and writing to a text file in VB2008 required the use of the StreamReader class and the StreamWriter class respectively. StreamReader is a tool that enables the streaming of data by moving it from one location to another so that it can be read by the user. For example, it allows the user to read a text file that is stored in a hard drive. On the other hand, the StreamWriter class is a tool that can write data input by the use to a storage device such as the hard drive. In order to read a file from the hard disk or any storage device, we need to use the StreamReader class. To achieve that, first of all we need to include the following statement in the program code:

Imports System.IO

This line has to precede the whole program code as it is higher in hierarchy than the StreamReader Class. In Fact, this is the concept of object oriented programming where StreamReader is part of the namespace System.IO . It has to be put on top of the whole program (i.e. above the Public Class Form 1 statement). The word import means we import the namesapce System.IO into the program. Once we have done that , we can declare a variable of the streamReader data type with the following statement:

```
Dim FileReader As StreamReader
```

If we don't include the Imports System.IO, we have to use the statement

```
Dim FileReader As IO.StreamReader
```

Each time we want to use the StreamReader class.

Now, start a new project and name it in whatever name you wish. Now, insert the OpenFileDialog control into the form because we will use it to read the file from the storage device. The default name of the OpenFileDialog control is OpenFileDialog1, you can use this name or you can rename it with a more meaningful name. The OpenFileDialog control will return a DialogResult value which can determine whether the user clicks the OK button or Cancel button . We will also insert a command button and change its displayed text to 'Open'. It will be used by the user to open and read a certain text file. The following statement will accomplish the task above.

Example:

```
Dim results As DialogResult
results = OpenFileDialog1.ShowDialog
If results = DialogResult.OK Then
'Code to be executed if OK button was clicked
Else
'Code to be executed if Cancel button was clicked
End If
End Sub
```

Next, we insert a textbox and set its Multiline property to true. It is used for displaying the text from a text file. In order to read the text file, we need to create a new instant of the streamReader and connect it to a text file with the following statement:

```
FileReader = New StreamReader(OpenFileDialog1.FileName)
```

In addition, we need to use the ReadToEnd method to read the entire text of a text file. The syntax is:

```
TextBox1.Text = FileReader.ReadToEnd()
```

Notes

Example:

Lastly, we need to close the file by using the Close() method. The entire code is shown in the box below:

```
Imports System.IO
Public Class Form1
Private Sub BtnOpen_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnOpen.Click
Dim FileReader As StreamReader
Dim results As DialogResult
results = OpenFileDialog1.ShowDialog
If results = DialogResult.OK Then
FileReader = New StreamReader(OpenFileDialog1.FileName)
TextBox1.Text = FileReader.ReadToEnd()
FileReader.Close()
End If
End Sub
```

Writing a text file: It means storing the text entered by the user via a textbox into a storage device such as a hard drive. It also means saving the file. To accomplish this task, we need to deploy the StreamWriter Class. You also need to insert the SaveFileDialog control into the form as it is used to save the data into the storage unit like a hard drive. The default bame for the SaveFileDialog control is SaveFileDialog1. The Code is basically the same as the code for reading the file, you just change the StreamReader to StreamWriter, and the method from ReadToEnd to Write.

Example:

```
Imports System.IO
Public Class Form1
Private Sub BtnSave_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
Dim FileWriter As StreamWriter
Dim results As DialogResult
results = SaveFileDialog1.ShowDialog
If results = DialogResult.OK Then
FileWriter = New StreamWriter(SaveFileDialog1.FileName, False)
FileWriter.Write(TextBox1.Text)
FileWriter.Close()
End If
End Sub
```

Self Assessment

True or False:

7. Directory class needs to be instantiated to be used.
8. DirectoryInfo class also has multiple constructors.
9. The FileInfo class is derived from the FileSystemInfo class.
10. The FileInfo class can be inherited.

Notes

11. VB.NET uses variable length records in order to implement random access files.
12. Notify Filters specifies changes to watch for in a file or folder.

12.3 System.IO Operations

The System.IO Namespace is one of the greatest time-savers in the .NET Framework. Not only is it a big time-saver, but it removes a lot of the nitty-gritty "bit-fiddling" involved with working with files, filenames, paths, etc.

To parse paths, filenames, and extensions in any version of VB prior to the advent of the .NET Framework was always a pain. Not only was it time-consuming, but it was fraught with errors.

```
Dim TestPath As String
        Dim nPos As Integer
        Dim sFileName As String
        Dim sFullPath As String
        Dim sExt As String
         TestPath = "C:\TESTPATH\TESTPATH2\FILENAME.TXT"
```

Example:

Here we extract only the path from a string which includes a path and a filename.

```
sFullPath = Path.GetDirectoryName(TestPath)
MsgBox(sFullPath)
```

Next, let us retrieve just the file extension from a path and filename.

```
sExt = Path.GetExtension(TestPath)
MsgBox(sExt)
```

Now, let us get the filename only from the path and filename.

```
 sFileName = Path.GetFileNameWithoutExtension(TestPath)
 MsgBox(sFileName)
```

To get the root directory from a path, then

```
 sRoot = Mid(TestPath, 1, InStr(TestPath, "\"))
 MsgBox(sRoot)
  ' get the root directory
sRoot = Path.GetPathRoot(TestPath)
MsgBox(sRoot)
```

Case Study

IO Virtualisation Technology Overview

The Atlantis Approach to IO Virtualisation

Atlantis ILIO is an IO virtualisation technology for VDI that front ends a traditional SAN/NAS storage system to enable high performance IO, storage optimization/consolidation and virtual desktop image composition.

Extracting IO State and Creating Flocks™

Flocks (File and Block IO) are Atlantis Computing patent pending format of virtualised data storage that merges file and block data into a component. Flocks are block devices

Contd...

Notes

with semantic knowledge of their contents. Flocks are aggregated and composited to form dynamic volumes that host NTFS and Unix filesystems. The benefit of storing data in flocks is that a common set of OS and application components can be used to create desktops and servers that are fully customisable (including allowing users to install applications) yet allow the underlying common OS and applications to be centrally changed (without the limitations suffered by traditional block based Copy on Write approaches like linked clones) or resource intensive instance-based file system based copy on write and kernel installed read write redirection approaches which do not guarantee application compatibility.

Real Time In-line De-duplication of Storage Objects

Traditional de-duplication is applied to a volume after duplicated data has been committed to it. Traditional de-duplication processes a volume by calculating a unique check sum for each block of data based on its contents and then sharing references to a single block wherever blocks have the same checksum. De-duplication has been used with virtualisation as a means to reduce the physical storage requirements for virtual machine images where the image content is highly repetitious. ILIO real time de-duplication works by marking and identifying duplicate objects as they are virtualised into flocks before data is committed to the storage volume. As a result the de-duplication process is more efficient and processes data in real time rather than as a post process. In practice this means that if two users on a VDI desktop pool install the same application, only one physical copy of the application is stored.

Using a Single Common Image of Operating System and Applications

ILIO allows the use of a single common physical copy of an operating system or application to be used across all virtual desktops instances. In practice this means that in a 10,000 instance Virtual Desktop implementation, there is just one copy of the OS and applications and all the 10,000 instances share the same base image. This typically results in a 20x storage reduction.

Caching Frequently Used IO's and Offloading the Backend SAN/NAS

Latency between SAN/NAS storage and VDI servers is a critical parameter in determining overall performance for a VDI deployment. As discussed previously, VDI is IO intensive and needs low latencies coupled with large throughput to work well. On loaded SAN/NAS systems latencies increase into the 10s of milliseconds and this can have negative consequences on user experience in the case of virtual desktops. ILIO allows frequently accessed and common IO to be cached close to the server. This serves two important purposes:

SAN/NAS Offload

ILIO can offload up to 95% of the IO on homogenous image deployment (as is the case with VDI where most OS and application components are shared). This means that a SAN/NAS system is no longer overwhelmed by many desktops making IO requests.

IO Acceleration

Atlantis ILIO allows important classes of IO such as Operating system virtual memory page files and application swap files to be served from cache thus dramatically improving the performance of IO intensive virtual workloads like virtual desktops.

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://www.atlantiscomputing.com/technology/io-virtualization-overview/

Notes

Self Assessment

Fill in the blanks:

13. To get the file extension from a path and filename we use the ..... method.
14. To get the filename only from the path and filename we use the ..... method.
15. To get the root directory use the ..... method.

12.4 Summary

- The Stream class is abstract; you can't declare a new instance of type Stream in your code.
- There are five classes in the .NET Framework that derive from the Stream class. These are:FileStream, MemoryStream, NetworkStream Namespace, CryptoStream and BufferedStream.
- A file is a collection of data stored in a disk with a specific name and a directory path.
- The stream is basically the sequence of bytes passing through the communication path.
- The File Class provides static methods for the creation, copying, deletion, moving, and opening of files, and aids in the creation of FileStream objects.
- The Path Class performs operations on String instances that contain file or directory path information.
- DirectoryInfo class also has a single constructor that takes either the full path or relative path to the directory as the input parameter.
- The FileInfo class is derived from the FileSystemInfo class and it cant be inherited.
- The File System Object (FSO) enables you to manipulate the files, folders and drives as well as read and write to sequential files.
- VB.NET uses fixed length records in order to implement random access files.
- Notify Filters specifies changes to watch for in a file or folder.

12.5 Keywords

BufferedStream: Supports buffered access to stream that do not support buffering on their own.

CryptoStream: Supports data encryption and decryption.

File: It is a collection of data stored in a disk with a specific name and a directory pat.

File Class: It provides static methods for the creation, copying, deletion, moving, and opening of files, and aids in the creation of FileStream objects.

FileStream: Supports sequential and random access to files.

MemoryStream: Supports sequential and random access to memory buffers.

NetworkStream: Supports sequential access to Internet resources.

Stream: It is basically the sequence of bytes passing through the communication path.

12.6 Review Questions

1. Differentiate between a file and directory.
2. What is a stream?
3. Explain the two types of stream.

Notes

4. What is the File Class?
5. Write a short note on the Basic File Class Operations.
6. What is a File System Object?
7. Explain Notify Filters with suitable examples.
8. Write a small note on the different types of streams.
9. Give the code to read and write in a text file.
10. Write a note on System.IO namespace.

Answers: Self Assessment

1.	Absract	2.	FileStream
3.	File	4.	Stream
5.	File	6.	Path
7.	False	8.	False
9.	True	10.	False
11.	False	12.	True
13.	GetExtension	14.	GetFileNameWithoutExtension
15.	GetPathRoot		

12.7 Further Readings

Books

Building Distributed Applications with Visual Basic.NET, Dan Fox, Sams.

Object-oriented Programming with Visual Basic.Net, Hamilton.

Programming Visual Basic .NET, J. Liberty.

VB .NET Language in a Nutshell, Steven Roman, Ron Petrusha, Paul Lomax.

Online links

http://www.java2s.com/Tutorial/VB/0240__StreamFile/FileCreationLastWriteandLastAccesstime.htm

http://www.devx.com/dotnet/Article/6971/0/page/2

http://www.techrepublic.com/article/working-with-net-files-via-the-path-class/6178071

http://www.yevol.com/en/vb/applicationdesign/Lesson38.htm

Notes

Unit 13: ADO.NET

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain database access using ADO.NET
- Understand insertion process
- Elaborate the execution, deletion and updation of data
- Discuss the select command with databases

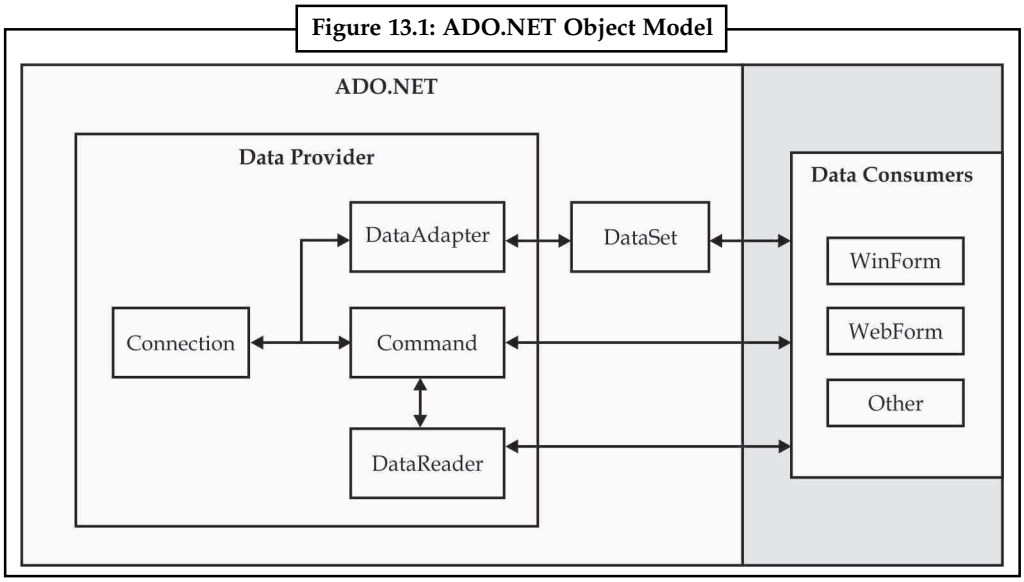
Introduction

ADO.NET provides consistent access to data sources such as Microsoft SQL Server, as well as data sources exposed through OLE DB and XML. Data-sharing consumer applications can use ADO.NET

Notes

to connect to these data sources and retrieve, manipulate, and update data. ADO.NET cleanly factors data access from data manipulation into discrete components that can be used separately or in tandem. ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, or placed in an ADO.NET **DataSet** object in order to be exposed to the user in an ad-hoc manner, combined with data from multiple sources, or remoted between tiers. The ADO.NET **DataSet** object can also be used independently of a .NET Framework data provider to manage data local to the application or sourced from XML. The ADO.NET classes are found in System.Data.dll, and are integrated with the XML classes found in System.Xml.dll. When compiling code that uses the **System.Data** namespace, reference both System.Data.dll and System.Xml.dll. ADO.NET provides functionality to developers writing managed code similar to the functionality provided to native COM developers by ADO.

Figure 13.1: ADO.NET Object Model



Source: http://www.codeproject.com/Articles/8477/Using-ADO-NET-for-beginners

13.1 Accessing Database with ADO.NET

ADO.NET is the new database technology of the .NET (Dot Net) platform, and it builds on Microsoft ActiveX Data Objects (ADO).ADO is a language-neutral object model that is the keystone of Microsoft's Universal Data Access strategy.ADO.NET is an integral part of the .NET Compact Framework, providing access to relational data, XML documents, and application data. ADO.NET supports a variety of development needs. You can create database-client applications and middle-tier business objects used by applications, tools, languages or Internet browsers. ADO.NET defines DataSet and DataTable objects which are optimized for moving disconnected sets of data across intranets and Internets, including through firewalls. It also includes the traditional Connection and Command objects, as well as an object called a DataReader that resembles a forward-only, read-only ADO recordset. If you create a new application, your application requires some form of data access most of the time. ADO.NET provides data access services in the Microsoft .NET platform.

You can use ADO.NET to access data by using the new .NET Framework data providers which are:

- Data Provider for SQL Server (System.Data.SqlClient).
- Data Provider for OLEDB (System.Data.OleDb).

Notes

- Data Provider for ODBC (System.Data.Odbc).
- Data Provider for Oracle (System.Data.OracleClient).

ADO.NET is a set of classes that expose data access services to the .NET developer. The ADO.NET classes are found in System.Data.dll and are integrated with the XML classes in System.Xml.dll.

There are two central components of ADO.NET classes: the DataSet, and the .NET Framework Data Provider.

Data Provider is a set of components including:

- Connection object (SqlConnection, OleDbConnection, OdbcConnection, OracleConnection)
- Command object (SqlCommand, OleDbCommand, OdbcCommand, OracleCommand)
- DataReader object (SqlDataReader, OleDbDataReader, OdbcDataReader, OracleDataReader)
- DataAdapter object (SqlDataAdapter, OleDbDataAdapter, OdbcDataAdapter, OracleDataAdapter).

DataSet object represents a disconnected cache of data which is made up of DataTables and DataRelations that represent the result of the command.

13.1.1 To Create the Database in MS Access

Follow the steps below to create a database in MS Access.

1. You may notice that, whenever you start Microsoft Access, you will see the screen and you can simply select "Blank Database".
2. If you already have Access open, you can go to the "File" menu and click on "New Database"
3. Choose "Blank Database". (Skip this step if you already chose "Blank Database" at step one). You also have the ability to choose from a template, but we'll just use a blank database.
4. Choose a location to save the database.
5. Once you've completed the above tasks, you should see a blank database.
6. We know this database is blank because it doesn't have any tables. If it did, you would see these tables in the middle pane of the table tab. Now that we have our blank database, we can start adding some tables.

13.1.2 Display the Database Records

OLE DB is a specification for wrapping data sources in a COM-based API so that data sources can be accessed in a polymorphic way. The concept is the same as ADO.NET's concept of managed providers. OLE DB predates ADO.NET and will eventually be superseded by it. However, over the years, OLE DB providers have been written for many data sources, including Oracle, Microsoft Access, Microsoft Exchange, and others, whereas currently only one product–SQL Server–is natively supported by an ADO.NET managed provider. To provide immediate support in ADO.NET for a wide range of data sources, Microsoft has supplied an ADO.NET managed provider for OLE DB. That means that ADO.NET can work with any data source for which there is an OLE DB data provider. Furthermore, because there is an OLE DB provider that wraps ODBC (an even older data-access technology), ADO.NET can work with virtually all legacy data, regardless of the source.

Notes

Connecting to an OLE DB data source is similar to connecting to SQL Server, with a few differences: the OleDbConnection class (from the System.Data.OleDb namespace) is used instead of the SqlConnection class, and the connection string is slightly different. When using the OleDbConnection class, the connection string must specify the OLE DB provider that is to be used as well as additional information that tells the OLE DB provider where the actual data is. For example, the following code opens a connection to the Northwind sample database in Microsoft Access:

‘ Open a connection to the database.

```
Dim strConnection As String = _
    "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" _
    & "C:\Program Files\Microsoft Office\Office\Samples\Northwind.mdb"
Dim cn As OleDbConnection = New OleDbConnection(strConnection)
cn.Open( )
```

Similarly, this code opens a connection to an Oracle database:

```
' Open a connection to the database.
Dim strConnection As String = _
    "Provider=MSDAORA.1;User ID=MyID;Password=MyPassword;" _
    & "Data Source=MyDatabaseService.MyDomain.com"
Dim cn As OleDbConnection = New OleDbConnection(strConnection)
cn.Open( )
```

The values of each setting in the connection string, and even the set of settings that are allowed in the connection string, are dependent on the specific OLE DB provider being used. Refer to the documentation for the specific OLE DB provider for more information.

Table 13.1 shows the provider names for several of the most common OLE DB providers.

Table 13.1: Common OLE DB Provider Names

Data source	OLE DB provider name
Microsoft Access	Microsoft.Jet.OLEDB.4.0
Microsoft Indexing Service	MSIDXS.1
Microsoft SQL Server	SQLOLEDB.1
Oracle	MSDAORA.1

Source: http://oreilly.com/catalog/progvbdotnet/chapter/ch08.html

The DataSet class is ADO.NET’s highly flexible, general-purpose mechanism for reading and updating data. Example below shows how to issue a SQL SELECT statement against the SQL Server Northwind sample database to retrieve and display the names of companies located in London.

Example: Retrieving data from SQL Server using a SQL SELECT statement

‘ Open a connection to the database.

```
Dim strConnection As String = _
    "Data Source=localhost; Initial Catalog=Northwind;" _
    & "Integrated Security=True"
Dim cn As SqlConnection = New SqlConnection(strConnection)
cn.Open( )

' Set up a data set command object.
```

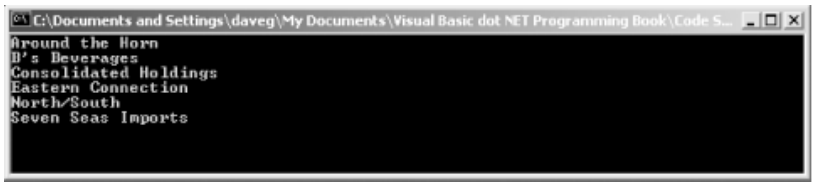
Notes

```
Dim strSelect As String = "SELECT * FROM Customers WHERE City = 'London'"
Dim dscmd As New SqlDataAdapter(strSelect, cn)

' Load a data set.
Dim ds As New DataSet( )
dscmd.Fill(ds, "LondonCustomers")

' Close the connection.
cn.Close( )

' Do something with the data set.
Dim dt As DataTable = ds.Tables.Item("LondonCustomers")
Dim rowCustomer As DataRow
For Each rowCustomer In dt.Rows
    Console.WriteLine(rowCustomer.Item("CompanyName"))
Next
```

The code above performs the following steps to obtain data from the database:

1. Opens a connection to the database using a SqlConnection object.
2. Instantiates an object of type SqlDataAdapter in preparation for filling a DataSet object. A SQL SELECT command string and a Connection object are passed to the SqlDataAdapter object's constructor.
3. Instantiates an object of type DataSet and fills it by calling the SqlDataAdapter object's Fill method.

Figure 13.2: Output of the Code Above



Source: http://oreilly.com/catalog/progvbdotnet/chapter/ch08.html

13.1.3 To Configure the Database

To get started running the 'unit test' you should configure the database connection string. The listing in DataQuickStart.GenericTemplate.ExampleTests.xml is shown below:

```
<objects xmlns="http://www.springframework.net"
             xmlns:db="http://www.springframework.net/database">

   <db:provider id="dbProvider"
                  provider="SqlServer-1.1"
                  connectionString="Data
Source=(local);Database=Northwind;User
ID=springqa;Password=springqa;Trusted_Connection=False"/>


   <! – other definitions not shown

</objects>
```

Notes

You should change the value of the provider element to correspond to you database and the connection string as appropriate. Please refer to the documentation on the DbProvider abstraction for details particular to your database configuration. You should also install the Northwind database, which is available for SqlServer 2005 from this download location. The minimal schema to support other database providers may be supported in the future.

13.1.4 To Create the DataGrid Edit Column

Websites often display thousands of data in a GridView in ASP.Net. Usually admin can view the registered users on the website, but when an admin wants to edit or delete any fraud or duplicate or damaged data from the table there is a method in GridView to edit, delete and update.

Source Code:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
      <title>Untitled Page</title>
      <style type="text/css">
.Gridview
{
font-family:Verdana;
font-size:10pt;
font-weight:normal;
color:black;

}
</style>
<script type="text/javascript">

</script>
</head>
<body>
      <form id="form1" runat="server">
<div>
      <asp:GridView ID="GridView1" runat="server"
AutoGenerateColumns="false" DataKeyNames="id"
OnPageIndexChanging="GridView1_PageIndexChanging"
OnRowCancelingEdit="GridView1_RowCancelingEdit"
OnRowDeleting="GridView1_RowDeleting"
OnRowEditing="GridView1_RowEditing"
OnRowUpdating="GridView1_RowUpdating">
     <Columns>
            <asp:BoundField DataField="id" HeaderText="S.No." />
            <asp:BoundField DataField="name" HeaderText="Name" />
            <asp:BoundField DataField="address" HeaderText="address" />
            <asp:BoundField DataField="country" HeaderText="Country" />
```

Notes

```
            <asp:CommandField ShowEditButton="true" />
            <asp:CommandField ShowDeleteButton="true" />
        </Columns>
    </asp:GridView>

    </div>
<div>
<asp:Label ID="lblresult" runat="server"></asp:Label>
</div>
    </form>
</body>
</html>
```

Design:

The design part will look as in the following image:

Figure 13.3: Design View

S.No.	Name	Address	Country		
Databound	Databound	Databound	Databound	Edit	Delete
Databound	Databound	Databound	Databound	Edit	Delete
Databound	Databound	Databound	Databound	Edit	Delete
Databound	Databound	Databound	Databound	Edit	Delete
Databound	Databound	Databound	Databound	Edit	Delete

Source: http://www.c-sharpcorner.com/UploadFile/9f0ae2/gridview-edit-delete-and-update-in-Asp-Net/

Code behind:

```
using System;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

public partial class _Default : System.Web.UI.Page
{
      private SqlConnection conn = new SqlConnection("Data
Source=NEHASHAMA;Integrated Security=true;Initial Catalog=rp");
            protected void Page_Load(object sender, EventArgs e)
         {
               if (!IsPostBack)
             {
                  gvbind();
```

Notes

```
            }
        }
         protected void gvbind()
        {
             conn.Open();
               SqlCommand cmd = new SqlCommand("Select * from detail",
conn);
               SqlDataAdapter da = new SqlDataAdapter(cmd);
              DataSet ds = new DataSet();
             da.Fill(ds);
             conn.Close();
              if (ds.Tables[0].Rows.Count > 0)
            {
                  GridView1.DataSource = ds;
                  GridView1.DataBind();
            }
            else
            {
                     ds.Tables[0].Rows.Add(ds.Tables[0].NewRow());
                  GridView1.DataSource = ds;
                  GridView1.DataBind();
                    int columncount = GridView1.Rows[0].Cells.Count;
                   GridView1.Rows[0].Cells.Clear();
                    GridView1.Rows[0].Cells.Add(new TableCell());
                    GridView1.Rows[0].Cells[0].ColumnSpan = columncount;
                   GridView1.Rows[0].Cells[0].Text = "No Records Found";
            }

        }

           protected void GridView1_RowDeleting(object sender,
GridViewDeleteEventArgs e)
        {
                GridViewRow row = (GridViewRow)GridView1.Rows[e.RowIndex];
               Label lbldeleteid = (Label)row.FindControl("lblID");
             conn.Open();
               SqlCommand cmd = new SqlCommand("delete FROM detail where
id='"+Convert.ToInt32(GridView1.DataKeys[e.RowIndex].Value.ToString())+"'",
conn);
              cmd.ExecuteNonQuery();
             conn.Close();
            gvbind();

        }
           protected void GridView1_RowEditing(object sender,
GridViewEditEventArgs e)
        {
               GridView1.EditIndex = e.NewEditIndex;
            gvbind();
        }
           protected void GridView1_RowUpdating(object sender,
GridViewUpdateEventArgs e)
```

Notes

```
        {
            int userid =
Convert.ToInt32(GridView1.DataKeys[e.RowIndex].Value.ToString());
              GridViewRow row = (GridViewRow)GridView1.Rows[e.RowIndex];
             Label lblID = (Label)row.FindControl("lblID");
              //TextBox txtname=(TextBox)gr.cell[].control[];
             TextBox textName = (TextBox)row.Cells[0].Controls[0];
             TextBox textadd = (TextBox)row.Cells[1].Controls[0];
             TextBox textc = (TextBox)row.Cells[2].Controls[0];
              //TextBox textadd = (TextBox)row.FindControl("txtadd");
              //TextBox textc = (TextBox)row.FindControl("txtc");
            GridView1.EditIndex = -1;
            conn.Open();
              //SqlCommand cmd = new SqlCommand("SELECT * FROM detail",
conn);
              SqlCommand cmd = new SqlCommand("update detail set
name='"+textName.Text+"',address='"+textadd.Text+"',country='"+textc.Text+"'where
id='"+userid+"'",conn);
             cmd.ExecuteNonQuery();
            conn.Close();
            gvbind();
             //GridView1.DataBind();
        }
           protected void GridView1_PageIndexChanging(object sender,
GridViewPageEventArgs e)
        {
              GridView1.PageIndex = e.NewPageIndex;
            gvbind();
        }
           protected void GridView1_RowCancelingEdit(object sender,
GridViewCancelEditEventArgs e)
        {
             GridView1.EditIndex = -1;
            gvbind();
        }
}
```

Save all or press "Ctrl+S" and hit "F5" to run the page, the page will look as in the following image:

Figure 13.4: Design View

S.No.	Name	Address	Country		
1	1	fgdf	Anneesfff	Edit	Delete

Source: http://www.c-sharpcorner.com/UploadFile/9f0ae2/gridview-edit-delete-and-update-in-Asp-Net/

Click on "Edit the GridView", it will display Textboxes in each cell as in the following image:

Figure 13.5: Design View

S.No.	Name	Address	Country	
1	1	fgdf	Anneesfff	Update Candel

Source: http://www.c-sharpcorner.com/UploadFile/9f0ae2/gridview-edit-delete-and-update-in-Ap-Net/

Notes

Edit the value(s) here and click on the Update link, it will update all the data or to remove it click on the "Delete" link above the image shown.

> *Note* One note will be helpful for you, while describing Columns in GridView if you are using a boundfield then create objects of the control using cells[index] in the rowupdateing event of GridView, but if you are using controls itself like Label or textboxes etc then use Fincontrol("stringid").

13.1.5 DataGrid Operations

We explain various operations in the gridview using one example.

STEP 1: Creating a DataBase Table

Here we will use an existing database, say. SampleDB, which has Customers Table and basically contains the following field columns:

```
CustomerID - PK
CompanyName
ContactName
ContactTitle
Address
Country
```

STEP 2: Setting Up the Connection String

```
        <connectionStrings>
                <add name="DBConnection" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\SampleDB.mdf;Integrated
Security=True;User Instance=True" providerName="System.Data.SqlClient"/>
        </connectionStrings>
```

STEP 3: Setting up the GUI

Just for the simplicity, we set up the GUI like this:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
      <title>GridView Data Manipulation</title>
</head>
<body>
      <form id="form1" runat="server">
     <div>
            <table cellpadding="0" cellspacing="0">
              <tr>
                     <td style="width: 100px; height: 19px;">
                        Company ID</td>
                     <td style="width: 100px; height: 19px;">
                        Company</td>
                     <td style="width: 100px; height: 19px;">
                       Name</td>
```

Notes

```
                    <td style="width: 100px; height: 19px;">
                      Title</td>
                    <td style="width: 100px; height: 19px;">
                      Address</td>
                    <td style="width: 100px; height: 19px;">
                      Country</td>
                </tr>
                <tr>
                    <td style="width: 100px">
                        <asp:TextBox ID="TextBox1" runat="server"/></td>
                    <td style="width: 100px">
                        <asp:TextBox ID="TextBox2" runat="server"/></td>
                    <td style="width: 100px">
                        <asp:TextBox ID="TextBox3" runat="server"/></td>
                    <td style="width: 100px">
                        <asp:TextBox ID="TextBox4" runat="server"/></td>
                    <td style="width: 100px">
                        <asp:TextBox ID="TextBox5" runat="server"/></td>
                    <td style="width: 100px">
                        <asp:TextBox ID="TextBox6" runat="server"/></td>
                    <td style="width: 100px">
                        <asp:Button ID="Button1" runat="server"
                                Text="Add New"
                                OnClick="Button1_Click" />
                    </td>
                </tr>
            </table>

              <asp:GridView ID="GridView1" runat="server"
                            AutoGenerateColumns="false"
                            ShowFooter="true">
            <Columns>
                <asp:BoundField DataField="CustomerID"
                                HeaderText="ID" ReadOnly="true"/>
                <asp:BoundField DataField="CompanyName"
                                HeaderText="Company"/>
                <asp:BoundField DataField="ContactName" HeaderText="Name"/>
                <asp:BoundField DataField="ContactTitle"
                                HeaderText="Title" />
                <asp:BoundField DataField="Address" HeaderText="Address"/>
                <asp:BoundField DataField="Country" HeaderText="Country"/>
            </Columns>
            </asp:GridView>
        </div>
        </form>
    </body>
    </html>
```

Note We have set the CustomerID field to ReadOnly so that the field cannot be edited.

STEP 4: Binding GridView with Data

Notes

The details about "Binding GridView with Data" are discussed in the previous example. Here are the code blocks for binding the GridView.

```
public partial class _Default : System.Web.UI.Page
{
      protected void Page_Load(object sender, EventArgs e)
     {
          if (!IsPostBack)
         {
              BindGridView();
         }
     }

      private string GetConnectionString()
     {
      System.Configuration.ConfigurationManager.ConnectionStrings["DBConnection"].
          ConnectionString;
     }

      #region Bind GridView
      private void BindGridView()
     {
           DataTable dt = new DataTable();
           SqlConnection connection = new SqlConnection(GetConnectionString());
         try
         {
              connection.Open();
                string sqlStatement = "SELECT Top(10)* FROM Customers";
                SqlCommand cmd = new SqlCommand(sqlStatement, connection);
                SqlDataAdapter sqlDa = new SqlDataAdapter(cmd);

                sqlDa.Fill(dt);
                if (dt.Rows.Count > 0)
               {
                   GridView1.DataSource = dt;
                   GridView1.DataBind();
               }
         }
            catch (System.Data.SqlClient.SqlException ex)
         {
                   string msg = "Fetch Error:";
                  msg += ex.Message;
                   throw new Exception(msg);
         }
          finally
         {
              connection.Close();
         }
     }
      #endregion
}
```

Notes

Now, we already know how to bind our GridView with data from database. So let's proceed on adding a new data in GridView.

13.1.6 To Add a Record to the DataGrid

As you have noticed in STEP 2, we have added six TextBox and a Button in the web form in order for us to type the information there and Insert them to the database. Now let's create a method for executing the Update or Insert.

Here are the code blocks for our Insert and Update method in the code behind:

```
#region Insert New or Update Record
      private void UpdateOrAddNewRecord(string ID, string Company, string
Name, string Title, string Address, string Country, bool isUpdate)
    {
          SqlConnection connection = new SqlConnection(GetConnectionString());
           string sqlStatement = string.Empty;

         if (!isUpdate)
        {
               sqlStatement = "INSERT INTO Customers"+
"(CustomerID,CompanyName,ContactName,ContactTitle,Address,Country)" +
"VALUES
(@CustomerID,@CompanyName,@ContactName,@ContactTitle,@Address,@Country)";
        }
        else
        {
              sqlStatement = "UPDATE Customers" +
                               "SET CompanyName = @CompanyName,
                               ContactName = @ContactName," +
                                "ContactTitle = @ContactTitle,Address =
                               @Address,Country = @Country" +
                                "WHERE CustomerID = @CustomerID,";
        }
        try
        {
              connection.Open();
                SqlCommand cmd = new SqlCommand(sqlStatement, connection);
                cmd.Parameters.AddWithValue("@CustomerID", ID);
                cmd.Parameters.AddWithValue("@CompanyName", Company);
                cmd.Parameters.AddWithValue("@ContactName", Name);
                cmd.Parameters.AddWithValue("@ContactTitle", Title);
                cmd.Parameters.AddWithValue("@Address", Address);
                cmd.Parameters.AddWithValue("@Country", Country);
               cmd.CommandType = CommandType.Text;
              cmd.ExecuteNonQuery();
        }
           catch (System.Data.SqlClient.SqlException ex)
        {
               string msg = "Insert/Update Error:";
             msg += ex.Message;
              throw new Exception(msg);
```

Notes

```
        }
         finally
        {
              connection.Close();
        }
    }
     #endregion
```

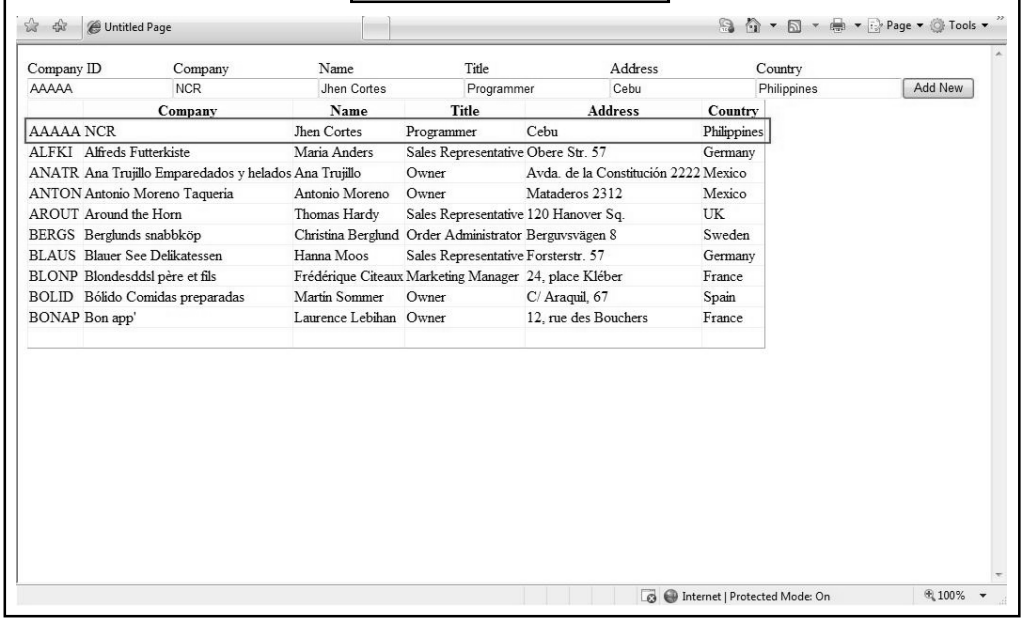
The UpdateOrAddNewRecord is a method that takes seven parameters. Six of those paramaters basically comes from the TextBox values that were entered in the page. The last parameter is a boolean value which tells the method whether to execute an Insert (false) or Update (true). Default is true.

Here's the code block for calling the method UpdateOrAddNewRecord on Button_Click event and pass the corresponding parameters needed:

```
      protected void Button1_Click(object sender, EventArgs e)
    {
            UpdateOrAddNewRecord(TextBox1.Text, TextBox2.Text,
TextBox3.Text, TextBox4.Text, TextBox5.Text, TextBox6.Text, false);
           //Re Bind GridView to reflect changes made
          BindGridView();
    }
```

As you can see from above, We have called the BindGridView() method again in order to reflect the changes made and display the new added data in the GridView. See output below:

Figure 13.6: Design View



Source: http://geekswithblogs.net/dotNETvinz/archive/2009/02/22/gridview-insert-edit-update-and-delete—the-ado.net-way.aspx

13.1.7 Edit and Update Records

One of the good things about GridView is that it provides a built-in CommandField Buttons which allows us to perform certain actions like editing, updating, deleting and selecting of GridView data.

Notes

To add those command fields mentioned in the GridView you can follow these few steps below:

1. Switch to Design View
2. Right Click on the GridView and Select —> Show Smart Tag —> Add New Columns
3. On the List Select CommandField
4. Check Delete and Edit/Update options then OK

As you can see the Edit and Delete CommandField are automatically added in the last column of GridView. Now we can start to write our codes for editing and updating the information in the GridView.

In-order to perform Edit and Update in GridView we need to use three events (GridView_RowEditing, GridView_RowCancelingEdit , GridView_RowUpdating). For those who do not know on how to generate Events in GridView you can follow the steps mentioned below:

1. Switch to Design View in Visual Studio Designer
2. Click on the GridView
3. Navigate to the GridView Property Pane and then SWITCH to Event Properties
4. From there you would be able to find the list of events including those three events mentioned above
5. Double Click on that to generate the Event handler for you
6. Then write the codes there

Here's the code for each events:

```
protected void GridView1_RowEditing(object sender, GridViewEditEventArgs
e)
{
            GridView1.EditIndex = e.NewEditIndex; // turn to edit mode
            BindGridView(); // Rebind GridView to show the data in edit mode
}

protected void GridView1_RowCancelingEdit(object sender,
GridViewCancelEditEventArgs e)
{
            GridView1.EditIndex = -1; //switch back to default mode
            BindGridView(); // Rebind GridView to show the data in default
mode
}

protected void GridView1_RowUpdating(object sender,
GridViewUpdateEventArgs e)
{
            //Accessing Edited values from the GridView
             string id = GridView1.Rows[e.RowIndex].Cells[0].Text; //ID
          string company =
((TextBox)GridView1.Rows[e.RowIndex].Cells[1].Controls[0]).Text; //
Company
          string name =
((TextBox)GridView1.Rows[e.RowIndex].Cells[2].Controls[0]).Text; //Name
          string title =
```

Notes

```
((TextBox)GridView1.Rows[e.RowIndex].Cells[3].Controls[0]).Text;  //Title
         string address =
((TextBox)GridView1.Rows[e.RowIndex].Cells[4].Controls[0]).Text;  //
Address
         string country =
((TextBox)GridView1.Rows[e.RowIndex].Cells[5].Controls[0]).Text;  //
Country

           UpdateOrAddNewRecord(id,company,name,title,address,country,true);
//  call  update  method
          GridView1.EditIndex  =  -1;
           BindGridView();  //  Rebind  GridView  to  reflect  changes  made
}
```

13.1.8 Delete Data in Gridview

Since we are using the Built-in Delete CommandField Button in GridView, we can use the GridView_RowDeleting event to delete specific row in GridView.

Here's the code block for the Delete method:

```
#region  Delete  Record
      private  void  DeleteRecord(string  ID)
    {
          SqlConnection  connection  =  new  SqlConnection(GetConnectionString());
           string  sqlStatement  =  "DELETE  FROM  Customers  WHERE  CustomerID  =
@CustomerID";
        try
        {
              connection.Open();
                SqlCommand  cmd  =  new  SqlCommand(sqlStatement,  connection);
                 cmd.Parameters.AddWithValue("@CustomerID",  ID);
               cmd.CommandType  =  CommandType.Text;
              cmd.ExecuteNonQuery();
        }
           catch  (System.Data.SqlClient.SqlException  ex)
        {
              string  msg  =  "Deletion  Error:";
             msg  +=  ex.Message;
              throw  new  Exception(msg);

        }
         finally
        {
              connection.Close();
        }
    }
#endregion
```

Here's the code block for calling the delete method at RowDeleting event

```
protected  void  GridView1_RowDeleting(object  sender,  GridViewDeleteEventArgs
e)
{
```

Notes

```
    string id = GridView1.Rows[e.RowIndex].Cells[0].Text; get the id of the selected row
    DeleteRecord(id);//call delete method
    BindGridView();//rebind grid to reflect changes made
}
```

Self Assessment

Fill in the blanks:

1. ..... provides consistent access to data sources.
2. The ADO.NET ..... object can also be used independently of a .NET Framework data provider to manage data local to the application or sourced from XML.
3. ADO is a ..... -neutral object model.
4. ADO.NET is a ..... architecture.
5. ..... that resembles a forward-only, read-only ADO recordset.
6. ADO.NET classes are found in ..... .
7. ..... represent the result of the command.

13.2 Executing Insertion

We have already discussed data insertion through ADO.NET in the section above.

13.2.1 Inserting Data Using ADO.NET

```
SqlDataAdapter.InsertCommand Property is used to insert data using ADO.NET.

Syntax: Public Property InsertCommand As SqlCommand
```

During Update, if this property is not set and primary key information is present in the DataSet, the InsertCommand can be generated automatically if you set the SelectCommand property and use the SqlCommandBuilder. Then, any additional commands that you do not set are generated by the SqlCommandBuilder. This generation logic requires key column information to be present in the DataSet. When InsertCommand is assigned to a previously created SqlCommand, the SqlCommand is not cloned. The InsertCommand maintains a reference to the previously created SqlCommand object. If execution of this command returns rows, these rows can be added to the DataSet depending on how you set the **UpdatedRowSource** property of the SqlCommand object.

For every column that you propagate to the data source on Update, a parameter should be added to InsertCommand, UpdateCommand, or DeleteCommand. The SourceColumn property of the parameter should be set to the name of the column. This indicates that the value of the parameter is not set manually, but is taken from the particular column in the currently processed row.

Example: The following example creates a SqlDataAdapter and sets the SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand properties. It assumes you have already created a SqlConnection object.

```
Public Function CreateCustomerAdapter( _
   ByVal connection As SqlConnection) As SqlDataAdapter
```

Notes

```
Dim adapter As SqlDataAdapter = New SqlDataAdapter()

' Create the SelectCommand.
Dim command As SqlCommand = New SqlCommand( _
    "SELECT * FROM Customers " & _
     "WHERE Country = @Country AND City = @City", connection)

' Add the parameters for the SelectCommand.
command.Parameters.Add("@Country", SqlDbType.NVarChar, 15)
command.Parameters.Add("@City", SqlDbType.NVarChar, 15)

adapter.SelectCommand = command

' Create the InsertCommand.
command = New SqlCommand( _
      "INSERT INTO Customers (CustomerID, CompanyName) " & _
       "VALUES (@CustomerID, @CompanyName)", connection)

' Add the parameters for the InsertCommand.
command.Parameters.Add("@CustomerID", SqlDbType.NChar, 5, "CustomerID")
 command.Parameters.Add("@CompanyName", SqlDbType.NVarChar, 40,
"CompanyName")

adapter.InsertCommand = command

' Create the UpdateCommand.
command = New SqlCommand( _
       "UPDATE Customers SET CustomerID = @CustomerID, CompanyName =
@CompanyName " & _
      "WHERE CustomerID = @oldCustomerID", connection)

' Add the parameters for the UpdateCommand.
command.Parameters.Add("@CustomerID", SqlDbType.NChar, 5, "CustomerID")
 command.Parameters.Add("@CompanyName", SqlDbType.NVarChar, 40,
"CompanyName")
Dim parameter As SqlParameter = command.Parameters.Add( _
      "@oldCustomerID", SqlDbType.NChar, 5, "CustomerID")
parameter.SourceVersion = DataRowVersion.Original

adapter.UpdateCommand = command

' Create the DeleteCommand.
command = New SqlCommand( _
     "DELETE FROM Customers WHERE CustomerID = @CustomerID", connection)

' Add the parameters for the DeleteCommand.
command.Parameters.Add( _
      "@CustomerID", SqlDbType.NChar, 5, "CustomerID")
parameter.SourceVersion = DataRowVersion.Original

adapter.DeleteCommand = command
```

Notes

```
    Return adapter
End Function
```

Self Assessment

True or False:

8. InsertCommand can be generated automatically if you set the SelectCommand property and use the SqlCommandBuilder.
9. When InsertCommand is assigned to a previously created SqlCommand, the SqlCommand is cloned.

13.3 Executing Deletion

Deleting a data in database can be possible with using DELETE command.

13.3.1 Deleting Data Using ADO.NET

Here we see how to use ADO.NET to connect to a SQL Server database, count the records and delete the records from the database. We create a table in SQL Server database which has the name emp3 and using count statement to count the row and delete command delete the records from table.

Creating Connection Object

To create a connection we pass the connection string as a parameter in connection object.

```
Dim str As String = "Data Source=.;uid=sa;pwd=123;database=master"
Dim con As New SqlConnection(str)
```

Count Rows in a Table

To count all the rows of the table we use the following statement.

```
select count(*) from emp3
```

Delete Command

The delete command is used to delete the record from database has the below command:

```
delete from emp3 where firstname='rahul' AND lastname='kumar'
```

Now we create a database table and insert some values in this table. Table looks like this.

Figure 13.7: Database Table

Results | Messages

	firstname	lastname
1	ram	kumar
2	ram	kumar
3	rahul	kumar
4	rahul	kumar
5	rahul	kumar
6	yatin	garg
7	monu	rathor

Source: http://www.dotnetheaven.com/article/ado.net-delete-command-in-vb.net

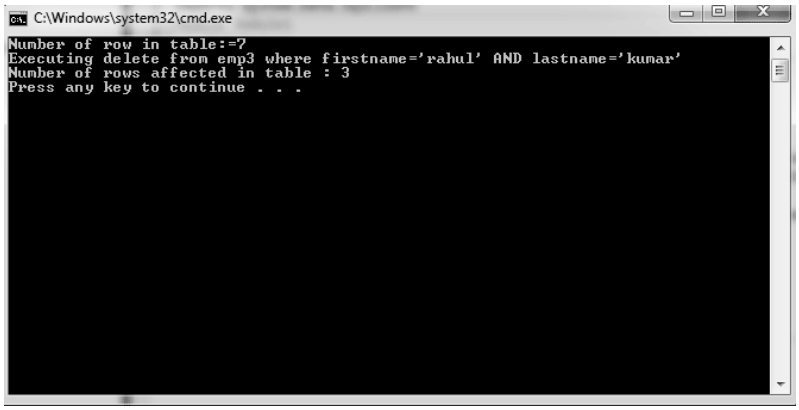
Notes

Example:

The below example defines the count command and delete command.

```
Imports System.Data.SqlClient
Module Module1
     Sub Main()
          Dim str As String = "Data
Source=.;uid=sa;pwd=123;database=master"
           Dim con As New SqlConnection(str)
         Try
              con.Open()
                Dim com As New SqlCommand("select count(*) from emp3", con)
                Console.WriteLine("Number of row in table:=" &
com.ExecuteScalar())
                 com.CommandText = "delete from emp3 where firstname='rahul'
AND lastname='kumar'"
                  Console.WriteLine("Executing {0}", com.CommandText)
                 Console.WriteLine("Number of rows affected in table : {0}",
com.ExecuteNonQuery())
              con.Close()
          Catch ex As Exception
                Console.WriteLine("can not delete record")
         End Try
     End Sub
End Module

Output:
```

Figure 13.8: Output of the Code Above



Source: http://www.dotnetheaven.com/article/ado.net-delete-command-in-vb.net

Now open the database and test it.

Self Assessment

True or False:

10. Deleting a data in database can be possible with using DELETE command.

11. To create a connection we pass the command object as a parameter in connection object

Notes

13.4 Executing Updation

The updation of data is done by UPDATE command.

13.4.1 Updating Data Using ADO.NET

We use the SqlDataAdapter.UpdateCommand Property to update data.

```
Syntax:
Public Property UpdateCommand As SqlCommand
```

During Update, if this property is not set and primary key information is present in the DataSet, the UpdateCommand can be generated automatically if you set the SelectCommand property and use the SqlCommandBuilder. Then, any additional commands that you do not set are generated by the SqlCommandBuilder. This generation logic requires key column information to be present in the DataSet.When UpdateCommand is assigned to a previously created SqlCommand, the SqlCommand is not cloned. The UpdateCommand maintains a reference to the previously created SqlCommand object.

Note If execution of this command returns rows, the updated rows may be merged with the DataSet depending on how you set the **UpdatedRowSource** property of the SqlCommand object.

For every column that you propagate to the data source on Update, a parameter should be added to InsertCommand, UpdateCommand, or DeleteCommand.

The SourceColumn property of the parameter should be set to the name of the column. This indicates that the value of the parameter is not set manually, but taken from the particular column in the currently processed row.

Example: This example creates a SqlDataAdapter and sets the SelectCommand, InsertCommand, UpdateCommand and DeleteCommand properties. It assumes you have already created a SqlConnection object.

```
Public Function CreateCustomerAdapter( _
   ByVal connection As SqlConnection) As SqlDataAdapter

      Dim adapter As SqlDataAdapter = New SqlDataAdapter()

      ' Create the SelectCommand.
      Dim command As SqlCommand = New SqlCommand( _
          "SELECT * FROM Customers " & _
           "WHERE Country = @Country AND City = @City", connection)

      ' Add the parameters for the SelectCommand.
      command.Parameters.Add("@Country", SqlDbType.NVarChar, 15)
      command.Parameters.Add("@City", SqlDbType.NVarChar, 15)

      adapter.SelectCommand = command

      ' Create the InsertCommand.
```

Notes

```
     command = New SqlCommand( _
           "INSERT INTO Customers (CustomerID, CompanyName) " & _
           "VALUES (@CustomerID, @CompanyName)", connection)

     ' Add the parameters for the InsertCommand.
     command.Parameters.Add("@CustomerID", SqlDbType.NChar, 5, "CustomerID")
      command.Parameters.Add("@CompanyName", SqlDbType.NVarChar, 40,
"CompanyName")

     adapter.InsertCommand = command

     ' Create the UpdateCommand.
     command = New SqlCommand( _
           "UPDATE Customers SET CustomerID = @CustomerID, CompanyName =
@CompanyName " & _
           "WHERE CustomerID = @oldCustomerID", connection)

     ' Add the parameters for the UpdateCommand.
     command.Parameters.Add("@CustomerID", SqlDbType.NChar, 5, "CustomerID")
      command.Parameters.Add("@CompanyName", SqlDbType.NVarChar, 40,
"CompanyName")
     Dim parameter As SqlParameter = command.Parameters.Add( _
           "@oldCustomerID", SqlDbType.NChar, 5, "CustomerID")
     parameter.SourceVersion = DataRowVersion.Original

     adapter.UpdateCommand = command

     ' Create the DeleteCommand.
     command = New SqlCommand( _
          "DELETE FROM Customers WHERE CustomerID = @CustomerID", connection)

     ' Add the parameters for the DeleteCommand.
     command.Parameters.Add( _
           "@CustomerID", SqlDbType.NChar, 5, "CustomerID")
     parameter.SourceVersion = DataRowVersion.Original

     adapter.DeleteCommand = command

     Return adapter
End Function
```

Self Assessment

Fill in the blanks:

12. The updation of data is done by ..... command.
13. UpdateCommand can be generated automatically if you set the SelectCommand property and use the ..... .

Notes

13.5 Select Command with Databases

The select operator can be used, among other things, to display a value. The select keyword uses the following syntax:

SELECT *What*

Based on this, to use it, where it is needed, type select followed by a number, a word, a string, or an expression. the item to display follows some of the same rules as print. One of the differences between print and select is that:

- Print is mostly used for testing a simple value, a string, or an expression. Therefore, it displays its results in a regular white window under a tab labeled messages. Print can be used with only one value.
- Select is the most regularly used SQL operator. We will see that it is used to retrieve records from a table. For this reason, select displays its results in an organized window made of categories called columns, under a tab labeled results. Select can be used with more than one value.

When you create a select statement, what is on the right side of select must be a value. Here is an example:

```
select 226.75;
```

Based on this definition, instead of just being a value, the thing on the right side of select must be able to produce a value. As we will see in the next sections, you can create algebraic operations on the right side of select. Unlike print, select can be used to display more than one value. The values must be separated by commas. Here is an example:

```
select n'hourly salary', 24.85
```

Because we mentioned that the thing on the right side must produce a result, you can as well use another **select** statement that it itself evaluates to a result. to distinguish the **select** sections, the second one should be included in parentheses. Here is an example:

```
select (select 448.25);
go
```

When one **select** statement is created after another, the second is referred to as nested.

Just as you can nest one **select** statement inside of another, you can also nest one statement in another statement that itself is nested. Here is an example:

```
SELECT (SELECT (SELECT 1350.75));
GO
SELECT This AS That
```

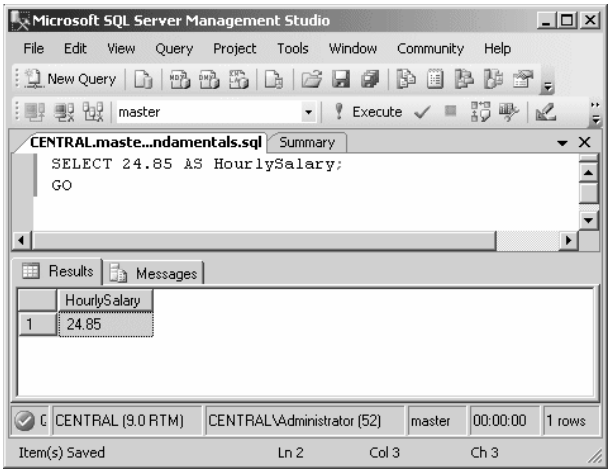
In the above introductions, we used either **PRINT** or **SELECT** to display something in the query window. One of the characteristics of **SELECT** is that it can segment its result in different sections. **SELECT** represents each value in a section called a column. Each column is represented with a name also called a caption. By default, the caption displays as "(No column name)". If you want to use your own caption, on the right side of an expression, type the **AS** keyword followed by the desired caption. The item on the right side of the **AS** keyword must be considered as one word. Here is an example:

```
SELECT 24.85 AS HourlySalary;
```

Notes

This would produce:

Figure 13.9: Output of the Select Statement



Source: http://www.functionx.com/vcsharp/adonet/select.htm

You can also include the item on the right side of **AS** in single-quotes. Here is an example:

```
SELECT 24.85 AS 'HourlySalary';
```

If the item on the right side of **AS** is in different words, you should include it in single-quotes or put them in inside of an opening square bracket "[" and a closing square bracket "]". Here is an example:

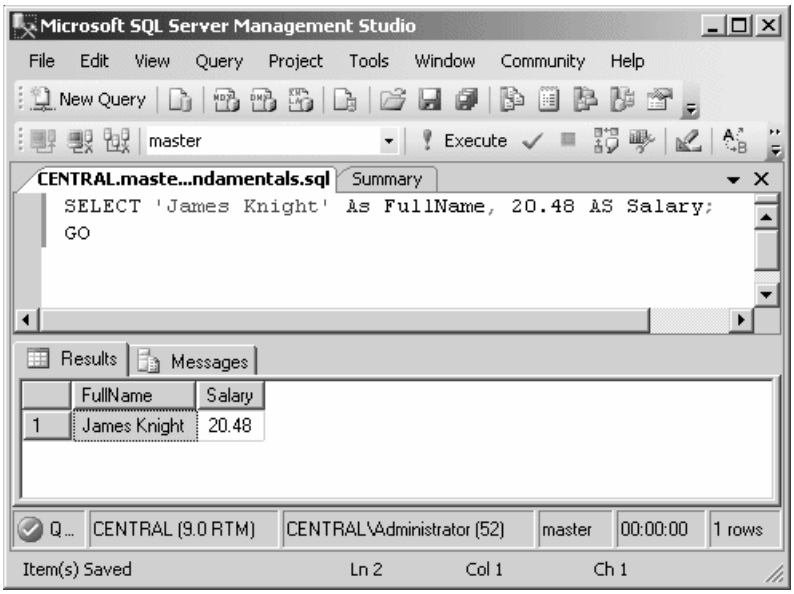
```
SELECT 24.85 AS 'Hourly Salary';
```

If you create different sections, separated by a comma, you can follow each with **AS** and a caption. Here is an example:

```
SELECT N'James Knight' As FullName, 20.48 AS Salary;
```

This would produce:

Figure 13.10: Output of the Select Statement



Source: http://www.functionx.com/vcsharp/adonet/select.htm

Notes

The above statement could also be written as follows:

```
SELECT N'James Knight' As [Full Name], 20.48 AS [Hourly Salary];
```

Case Study

Open Investor Relationship Management System

The Challenge

DataArt has developed an Open Investor Relationship Management System for a New York-based asset management firm. The system provides classic CRM functions as well as features specific to the operations of a financial services company. The following functional areas are supported by the system:

- Contact management
- Investment tracking
- Fund performance management
- Front-office research support
- Investor reporting

Solution

The system is entirely web-based, built on Microsoft's .NET technology platform. It is implemented in classic 3-tier architecture, providing good support for scalability and maintainability. SQL Server 2000 is used as the data storage layer, which communicates with the Data Management and Business Logic layers through ADO.NET. The data Management and Business Logic layers are organized as a set of Class Libraries written in C#, operating under the control of Application Server (IIS+.NET Framework). The top level, the User Interface, is based on the Microsoft IE 6.0/HTML/JavaScript platform to combine mobility gained through fully web-based interface with the advanced controls and user interaction methods provided by the IE 6.0 environment. Communication between the User Interface and Application Server is organized in two ways:

- Standard HTTP form-based data exchange is based on request-response roundtrips
- A Web Services-based protocol, utilizing IE's Web Services scenarios (.HTC behavior script). This options eliminates extra roundtrips to the server and thus intensifies the user experience The system maintains Web Services-based interfaces with a number of external systems, including the corporate website and the Microsoft Exchange server.

The .NET framework has brought a number of important technological benefits to the solution:

- Web Server Controls are used to increase reusability of the code
- Built-in support for caching is utilized extensively to minimize system response times and optimize load
- The .NET-based system proves to be exceptionally reliable and robust. Throughout a series of intensive tests, the number of system failures, memory leaks and the like appeared to be drastically less than in systems built on older platforms. Within this system, only 'safe' code is used, which has become possible only in the .NET environment

Contd...

Notes

- .NET's control over Impersonation allows the system to utilize different approaches for secure access rights control. For example, application-level access rights are used for normal user operations, while object-based restrictions, previously only typical for thick client architectures, are used for system administration tasks.

Tools and Technologies

MS SQL Server, ASP.NET, C#, XML, Microsoft HTC, Web Services

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://www.russoft.org/success/?story=143

Self Assessment

True or False:

14. Select can be used with only one value.
15. Print displays its results in an organized window made of categories called columns.

13.6 Summary

- ADO.NET provides consistent access to data sources.
- The ADO.NET DataSet object can also be used independently of a .NET Framework data provider to manage data local to the application or sourced from XML.
- ADO is a language-neutral object model.
- ADO.NET defines DataSet and DataTable objects which are optimized for moving disconnected sets of data across intranets and Internet.
- DataReader that resembles a forward-only, read-only ADO recordset.
- ADO.NET classes are found in System.Data.dll and are integrated with the XML classes in System.Xml.dll.
- DataSet object represents a disconnected cache of data which is made up of DataTables.
- DataRelations that represent the result of the command.
- InsertCommand can be generated automatically if you set the SelectCommand property and use the SqlCommandBuilder.
- When InsertCommand is assigned to a previously created SqlCommand, the SqlCommand is not cloned.
- Deleting a data in database can be possible with using DELETE command.
- To create a connection we pass the connection string as a parameter in connection object.
- The updation of data is done by UPDATE command.
- UpdateCommand can be generated automatically if you set the SelectCommand property and use the SqlCommandBuilder.

Notes

- The select operator can be used, among other things, to display a value.
- Print can be used with only one value.
- Select displays its results in an organized window made of categories called columns.

13.7 Keywords

ADO.NET DataSet object: It can also be used independently of a .NET Framework data provider to manage data local to the application or sourced from XML

ADO.NET: It provides consistent access to data sources

DataReader: It resembles a forward-only, read-only ADO recordset.

DataRelations: It represent the result of the command.

DataSet object: It represents a disconnected cache of data which is made up of DataTables

Delete command: It is used to delete data from a database through ADO.NET.

InsertCommand: It is used to insert data to a database through ADO.NET

UpdateCommand: It is used to update data from a database through ADO.NET.

13.8 Review Questions

1. What is ADO.NET?
2. Why do we use ADO.NET?
3. How do we access a database with ADO.NET?
4. Give the steps to display the Database Records using ADO.NET.
5. What do you mean by Configuring of the Database?
6. With the help of an example show how to Create the DataGrid Edit Column.
7. How do you Edit the DataGrid?
8. Write the code to update the DataGrid.
9. Explain the steps to delete data from the DataGrid.
10. Write short note on Insert, Select, Delete and Update commands.

Answers: Self Assessment

1.	ADO.NET	2.	DataSet
3.	Language	4.	Disconnected
5.	DataReader	6.	System.Data,dll
7.	DataRelations	8.	True
9.	False	10.	True
11.	False	12.	Update
13.	SQLCommandBuilder	14.	False
15.	False		

13.9 Further Readings

Notes

Books

Beginning Vb.Net 2003, Willis.

Building Distributed Applications with Visual Basic.NET, Dan Fox, Sams.

Object-oriented Programming with Visual Basic.Net, Hamilton.

VB .NET Language in a Nutshell, Steven Roman, Ron Petrusha, Paul Lomax.

Online links

http://msdn.microsoft.com/en-us/library/ms971485.aspx

http://www.4guysfromrolla.com/webtech/chapters/ASPNET/ch06.html

http://springframework.net/doc-latest/reference/html/data-quickstart.html

http://www.functionx.com/vcsharp/adonet/select.htm

Notes

Unit 14: XML

CONTENTS

Objectives

After studying this unit, you will be able to:

- Describe XML representation
- Explain data representation through XML
- Understand working with XML Reader and XML Writer Classes
- Discuss the advantages and disadvantages of XML

Introduction

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards.

XML has been widely adopted since its creation and with good reason. Some of the key features and benefits of XML include:

- ***Easy data exchange:*** One of the great things about XML is that it can allow easy sharing of data between different applications – even if these applications are written in different languages and reside on different platforms.
- ***Self-describing data:*** When you look at an XML document, it is very easy to figure out what's going on.

Notes

- ***Create your own languages:*** XML allows you to specify your own markup language for your own specific purpose. Some existing XML based languages include Banking Industry Technology Secretariat (BITS), Bank Internet Payment System (BIPS), Financial Exchange (IFX) and many more.

The basic difference between HTML and XML is that HTML was designed to display data and to focus on how data looks while XML is designed to describe data and to focus on what data is.

14.1 Representation of the XML

XML has a variety of uses for Web, e-business, and portable applications. The following are some of the many applications for which XML is useful:

- ***Web publishing:*** XML allows you to create interactive pages, allows the customer to customize those pages, and makes creating e-commerce applications more intuitive. With XML, you store the data once and then render that content for different viewers or devices based on style sheet processing using an Extensible Style Language (XSL)/XSL Transformation (XSLT) processor.
- ***Web searching and automating Web tasks:*** XML defines the type of information contained in a document, making it easier to return useful results when searching the Web.

Example: Using HTML to search for books authored by Tom Brown is likely to return instances of the term 'brown' outside of the context of author. Using XML restricts the search to the correct context (for example, the information contained in the <author> tag) and returns only the information that you want. By using XML, Web agents and robots (programs that automate Web searches or other tasks) are more efficient and produce more useful results.

- ***General applications:*** XML provides a standard method to access information, making it easier for applications and devices of all kinds to use, store, transmit, and display data.
- ***e-business applications:*** XML implementations make Electronic Data Interchange (EDI) more accessible for information interchange, business-to-business transactions, and business-to-consumer transactions.
- ***Metadata applications:*** XML makes it easier to express metadata in a portable, reusable format.
- ***Pervasive computing:*** XML provides portable and structured information types for display on pervasive (wireless) computing devices such as Personal Digital Assistants (PDAs), cellular phones, and others.

Example: WML (Wireless Markup Language) and VoiceXML are currently evolving standards for describing visual and speech-driven wireless device interfaces.

A small example just to show what an XML file looks like is given below:

```
<?xml version="1.0" encoding="UTF-8"?>
<tutorials>
   <tutorial>
      <name>XML Tutorial</name>
      <url>http://www.abc.com</url>
   </tutorial>
   <tutorial>
      <name>HTML Tutorial</name>
```

Notes

```
        <url>http://www.abc.com</url>
    </tutorial>
</tutorials>
```

14.1.1 XML Schemas

Technically, a schema is an abstract collection of metadata, consisting of a set of schema components: chiefly element and attribute declarations and complex and simple type definitions. These components are usually created by processing a collection of schema documents, which contain the source language definitions of these components. In popular usage, however, a schema document is often referred to as a schema. Schema documents are organized by namespace: all the named schema components belong to a target namespace, and the target namespace is a property of the schema document as a whole. A schema document may include other schema documents for the same namespace, and may import schema documents for a different namespace. When an instance document is validated against a schema (a process known as assessment), the schema to be used for validation can either be supplied as a parameter to the validation engine, or it can be referenced directly from the instance document using two special attributes, xsi:schemaLocation and xsi:noNamespaceSchemaLocation. XML Schema Documents usually have the filename extension ".xsd". A unique Internet Media Type is not yet registered for XSDs, so "application/xml" or "text/xml" should be used, as per RFC 3023.

The main components of a schema are:

- Element declarations, which define properties of elements. These include the element name and target namespace. An important property is the type of the element, which constrains what attributes and children the element can have. In XSD 1.1, the type of the element may be conditional on the values of its attributes. An element may belong to a substitution group; if element E is in the substitution group of element H, then wherever the schema permits H to appear, E may appear in its place. Elements may have integrity constraints: uniqueness constraints determining that particular values must be unique within the subtree rooted at an element, and referential constraints determining that values must match the identifier of some other element. Element declarations may be global or local, allowing the same name to be used for unrelated elements in different parts of an instance document.
- Attribute declarations, which define properties of attributes. Again the properties include the attribute name and target namespace. The attribute type constrains the values that the attribute may take. An attribute declaration may also include a default value or a fixed value (which is then the only value the attribute may take.)
- Simple and complex types.
- Model group and attribute group definitions. These are essentially macros: named groups of elements and attributes that can be reused in many different type definitions.
- An attribute use represents the relationship of a complex type and an attribute declaration, and indicates whether the attribute is mandatory or optional when it is used in that type.
- An element particle similarly represents the relationship of a complex type and an element declaration, and indicates the minimum and maximum number of times the element may appear in the content. As well as element particles, content models can include model group particles, which act like non-terminals in a grammar: they define the choice and repetition units within the sequence of permitted elements. In addition, wildcard particles are allowed, which permit a set of different elements (perhaps any element provided it is in a certain namespace).

Notes

Other more specialized components include annotations, assertions, notations, and the schema component which contains information about the schema as a whole.

Complex types describe the permitted content of an element, including its element and text children and its attributes. A complex type definition consists of a set of attribute uses and a content model. Varieties of content model include element-only content, in which no text may appear (other than whitespace, or text enclosed by a child element); simple content, in which text is allowed but child elements are not; empty content, in which neither text nor child elements are allowed; and mixed content, which permits both elements and text to appear. A complex type can be derived from another complex type by restriction (disallowing some elements, attributes, or values that the base type permits) or by extension (allowing additional attributes and elements to appear). In XSD 1.1, a complex type may be constrained by assertions–XPath 2.0 expressions evaluated against the content that must evaluate to true. Simple types (also called data types) constrain the textual values that may appear in an element or attribute. This is one of the more significant ways in which XML Schema differs from DTDs. For example, an attribute might be constrained to hold only a valid date or a decimal number. XSD provides a set of 19 primitive data types (anyURI, base64Binary, boolean, date, dateTime, decimal, double, duration, float, hexBinary, gDay, gMonth, gMonthDay, gYear, gYearMonth, NOTATION, QName, string, and time). It allows new data types to be constructed from these primitives by three mechanisms:

- restriction (reducing the set of permitted values),
- list (allowing a sequence of values), and
- union (allowing a choice of values from several types).

Twenty-five derived types are defined within the specification itself, and further derived types can be defined by users in their own schemas.

The mechanisms available for restricting data types include the ability to specify minimum and maximum values, regular expressions, constraints on the length of strings, and constraints on the number of digits in decimal values. XSD 1.1 again adds assertions, the ability to specify an arbitrary constraint by means of an XPath 2.0 expression.

14.1.2 Document Type Definition (DTD)

A Document Type Definition (DTD) is a set of markup declarations that define a document type for an SGML-family markup language (SGML, XML, HTML). A DTD uses a terse formal syntax that declares precisely which elements and references may appear where in the document of the particular type, and what the elements' contents and attributes are. A DTD can also declare entities which may be used in the instance document. XML uses a subset of SGML DTD.A DTD is associated with an XML or SGML document by means of a Document Type Declaration. The Document Type Declaration appears in the syntactic fragment doctypedecl near the start of an XML document. The declaration establishes that the document is an instance of the type defined by the referenced DTD.

DTDs make two sorts of declaration:

- An optional external subset
- An optional internal subset

The declarations in the internal subset form part of the Document Type Declaration in the document itself. The declarations in the external subset are located in a separate text file. The external subset may be referenced via a public identifier and/or a system identifier. Programs for reading documents may not be required to read the external subset.

Notes

Note Any valid SGML or XML document that references an external subset in its DTD, or whose body contains references to parsed external entities declared in its DTD (including those declared within its internal subset), may only be partially parsed but cannot be fully validated by validating SGML or XML parsers in their standalone mode (this means that these validating parsers will not attempt to retrieve these external entities, and their replacement text will not be accessible).

The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements. A DTD can be declared inline in the XML document, or as an external reference. XML provides an application independent way of sharing data. With a DTD, the application can use a standard DTD to verify that data that the user supplies is valid. A "Valid" XML document is a "Well Formed" XML document which conforms to the rules of a Document Type Definition (DTD).

Example: A *drive* resource containing a *device* property and a *partitions* property represented as a nested resource. A *partitions* resource containing multiple instances of the *partition* property represented as a nested resource. A *partition* resource containing a *size* property and a *mount* property.

Below is the XML for an example node view profile for the above tree which includes a DTD which validates it.

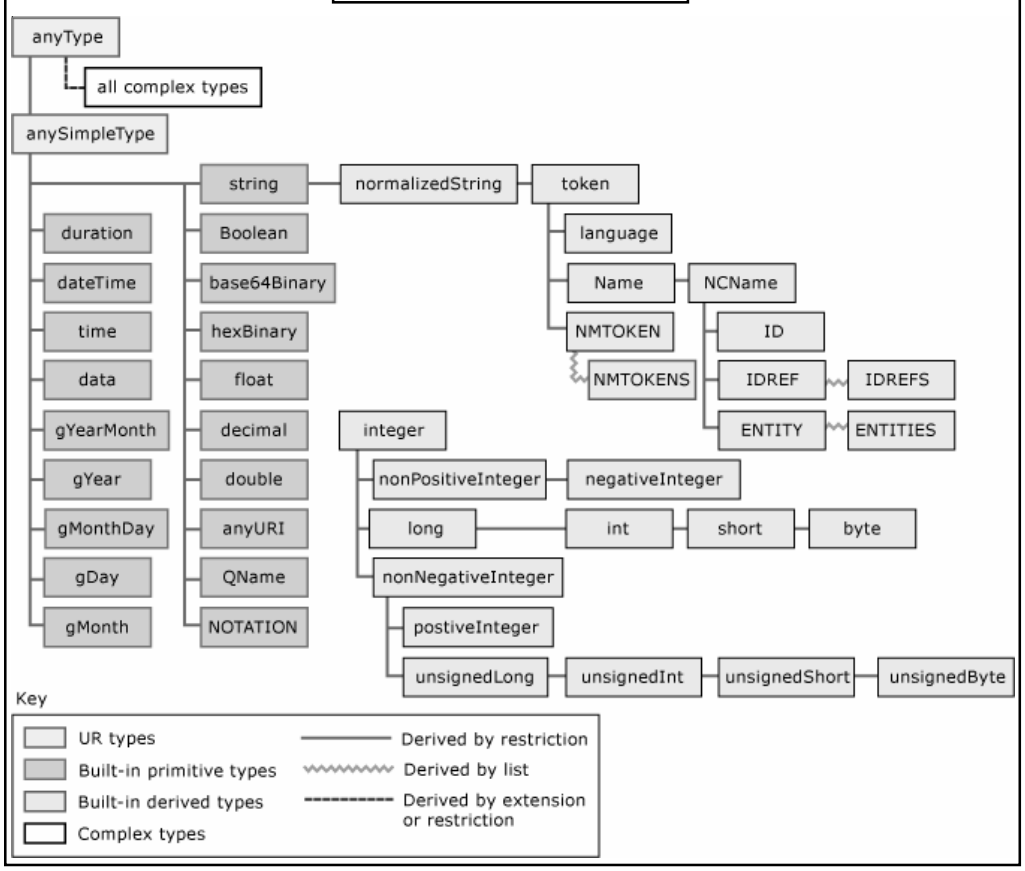
```
<?xml version="1.0"?>
<!DOCTYPE profile [
<!ELEMENT profile (install)>
<!ELEMENT install (partitioning)>
<!ELEMENT partitioning (drive+)>
<!ELEMENT drive (device,partitions)>
<!ELEMENT device (#PCDATA)>
<!ELEMENT partitions (partition*)>
<!ELEMENT partition (size,mount)>
<!ELEMENT size (#PCDATA)>
<!ELEMENT mount (#PCDATA)>
]>
<profile>
.....
   <install>
      <partitioning config:type="list">
       <drive>
         <device>
            /dev/hda
         </device>
         <partitions>
           <partition>
              <size>1000mb</size>
              <mount>/</mount>
           </partition>
           <partition>
              <size>250mb</size>
              <mount>/tmp</mount>
           </partition>
```

Notes

```
        </partitions>
      </drive>
     </partitioning>
   </install>
.....
</profile>
```

Datatypes: We will now discuss the basic types supported by XML.

Figure 14.1: Datatypes in XML



Source: http://msdn.microsoft.com/en-us/library/ms256131.aspx

- *Primitive data types:* The following section lists primitive XML schema data types, facets that can be applied to the data type, and a description of the data type. Facets can only appear once in a type definition except for **enumeration** and **pattern** facets. **Enumeration** and **pattern** facets can have multiple entries and are grouped together.

Example:

string - Represents character strings.

boolean - Represents Boolean values, which are either **true** or **false**.

decimal - Represents arbitrary precision numbers.

float - Represents single-precision 32-bit floating-point numbers.

double - Represents double-precision 64-bit floating-point numbers.

Notes

duration - Represents a duration of time.

dateTime - Represents a specific instance of time.

time - Represents an instance of time that recurs every day. The pattern for **time** is hh:mm:ss.sss with optional time zone indicator.

Date **-** Represents a calendar date. The pattern for **date** is CCYY-MM-DD with optional time zone indicator as allowed for **dateTime**.

anyURI - Represents a URI as defined by RFC 2396. An **anyURI** value can be absolute or relative, and may have an optional fragment identifier.

- ***Derived data types:*** The following section illustrates the derived XML schema data types, facets that can be applied to the derived data type, and a description of the derived data type.

Example:

token - Represents tokenized strings. This data type is derived from **normalizedString**.

language - Represents natural language identifiers (defined by RFC 1766). This data type is derived from **token**.

IDREFS - Represents the **IDREFS** attribute type. Contains a set of values of type **IDREF**.

ENTITIES - Represents the **ENTITIES** attribute type. Contains a set of values of type **ENTITY**.

Name - Represents names in XML. A **Name** is a token that begins with a letter, underscore, or colon and continues with name characters (letters, digits, and other characters). This data type is derived from **token**.

ID - Represents the **ID** attribute type defined in the XML 1.0 Recommendation. The **ID** must be a no-colon-name (NCName) and must be unique within an XML document. This data type is derived from **NCName**.

ENTITY - Represents the **ENTITY** attribute type in XML 1.0 Recommendation. This is a reference to an unparsed entity with a name that matches the specified name. An **ENTITY** must be an NCName and must be declared in the schema as an unparsed entity name. This data type is derived from **NCName**.

integer - Represents a sequence of decimal digits with an optional leading sign (+ or -). This data type is derived from **decimal**.

long - Represents an integer with a minimum value of -9223372036854775808 and maximum of 9223372036854775807. This data type is derived from **integer**.

short - Represents an integer with a minimum value of -32768 and maximum of 32767. This data type is derived from **int**.

byte - Represents an integer with a minimum value of -128 and maximum of 127. This data type is derived from **short**.

14.1.3 Characteristics of XML

XML has a number of important characteristics. Some of them are given below:

- ***XML is a structured format:*** Which means that we can define exactly how the data is to be arranged, organized and expressed within the file. When we are given a file, we can validate that it conforms to a specific structure, prior to importing the data. As we know

Notes

the structure of the file in advance, we know what it contains and how to process each item. Prior to XML, the only structure in a text file was positional – we knew the bit of text after the fourth comma should be a date of birth – and we had no way to validate whether it was a date of birth, or even a date, or whether it was in day/month/year or month/day/year order.

- ***XML is a described format:*** Which means that within the text file, every item of data has a name that is both human- and machine-readable as well as being uniquely identifiable. We can open these files, read their contents and understand the data they contain, without having to refer back to another document to find out what the text after the fourth comma represents (and was that comma a separator, or part of the text of the second item?). Similarly, we can edit these documents with a fairly high level of confidence that we're making the correct changes.
- XML can easily describe **hierarchical** data and the **relationships** between data. If we want to import and export a list of authors, with their names, addresses and the books they've written, deciding on a reasonable format for a csv file is by no means straightforward. Using XML, we can define what an Author item is and that it has a name, address and multiple Book items. We can also define what a Book item it is and that it has a title, a publisher and an ISBN. The hierarchy and relationships are a natural consequence of the definition.
- ***XML can be validated:*** Which means we can provide a second XML file – an XML Schema Definition file – that describes exactly how the XML data file should be structured. Before processing an XML file, we can compare it with the schema to ensure it conforms to the structure we expect to receive.
- ***XML is a discoverable format:*** Which means programs (including Excel 2003/2007/2010/2013) can parse an XML data file and infer the structure and relationships between the items. This means we can read an XML file, infer its structure and generate new XML data files that conform to the same structure, with a high degree of confidence the new XML data files will pass validation.
- ***XML is a strongly-typed format:*** Which means the schema definition file specifies the data type of each element. When importing the data, the application can check the schema definition to identify the data type to import it as. We no longer run the risk of the product code 01-03 being imported as a date.
- ***XML is a global format:*** There is only one way to express a number in an XML file (with US number formats) and only one way to express a date. We no longer have to check whether a csv file was created with US or French settings and adjust our processing of it accordingly.
- ***XML is a standard format:*** The way in which the content of an XML file is defined has been specified by the World Wide Web Consortium (W3C). This allows applications (including Excel 2003/2007/2010/2013) to read, understand and validate the structure of an XML file and create files that conform to the specified structure. It also allows different applications to read, write, understand and validate the same XML files, allowing us to share data between applications in an extremely robust manner.

Self Assessment

Fill in the blanks:

1. ..... is a markup language that defines a set of rules for encoding documents schema is an abstract collection of metadata.

Notes

2. Schema documents are organized by .....
3. XML Schema Documents usually have the filename extension .....
4. ..... declarations, which define properties of elements.
5. ..... declarations, which define properties of attributes
6. ..... derived types are defined within the specification itself.
7. The purpose of a ..... is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements.

14.2 Data Representation through XML

XML documents form a tree structure that starts at "the root" and branches to "the leaves".

Example:

XML documents use a self-describing and simple syntax:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
   <to>Tove</to>
   <from>Jani</from>
   <heading>Reminder</heading>
   <body>Don't forget me this weekend!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set).

The next line describes the **root element** of the document (like saying: "this document is a note"):

```
<note>
```

The next 4 lines describe 4 **child elements** of the root (to, from, heading, and body):

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

And finally the last line defines the end of the root element:

```
</note>
```

You can assume, from this example, that the XML document contains a note to Tove from Jani.

XML documents must contain a **root element**. This element is "the parent" of all other elements. The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

All elements can have sub elements (child elements):

```
<root>
   <child>
      <subchild>.....</subchild>
   </child>
</root>
```

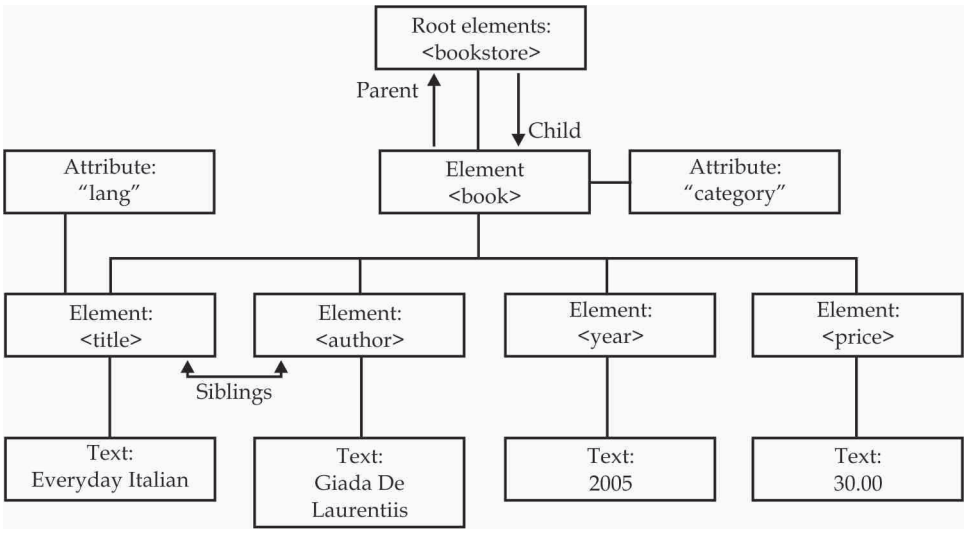
The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

All elements can have text content and attributes (just like in HTML).

Notes

Example:

Figure 14.2: XML Tree Structure



Source: http://www.w3schools.com/xml/xml_tree.asp

The image above represents one book in the XML below:

```
<bookstore>
   <book category="COOKING">
      <title lang="en">Everyday Italian</title>
      <author>Giada De Laurentiis</author>
      <year>2005</year>
      <price>30.00</price>
   </book>
   <book category="CHILDREN">
      <title lang="en">Harry Potter</title>
      <author>J K. Rowling</author>
      <year>2005</year>
      <price>29.99</price>
   </book>
   <book category="WEB">
      <title lang="en">Learning XML</title>
      <author>Erik T. Ray</author>
      <year>2003</year>
      <price>39.95</price>
   </book>
</bookstore>
```

The root element in the example is <bookstore>. All <book> elements in the document are contained within <bookstore>.

```
The <book> element has 4 children: <title>,< author>, <year>, <price>.
```

Some characters have a special meaning in XML.

Notes

Entity Reference: If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element. This will generate an XML error:

```
<message>if salary < 1000 then</message>
```

To avoid this error, replace the "<" character with an **entity reference**:

<message>if salary < 1000 then</message>

There are 5 predefined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

Note Only the characters "<" and "&" are strictly illegal in XML. The greater than character is legal, but it is a good habit to replace it.

Comments in XML: The syntax for writing comments in XML is similar to that of HTML.

```
<!— This is a comment —>
```

XML Elements: An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain:

- other elements
- text
- attributes
- or a mix of all of the above...

```
<bookstore>
   <book category="CHILDREN">
      <title>Harry Potter</title>
      <author>J K. Rowling</author>
      <year>2005</year>
      <price>29.99</price>
   </book>
   <book category="WEB">
      <title>Learning XML</title>
      <author>Erik T. Ray</author>
      <year>2003</year>
      <price>39.95</price>
   </book>
</bookstore>
```

In the example above, <bookstore> and <book> have **element contents**, because they contain other elements. <book> also has an **attribute** (category="CHILDREN"). <title>, <author>, <year>, and <price> have **text content** because they contain text.

Notes

XML Attributes: XML elements can have attributes, just like HTML.Attributes provide additional information about an element. In HTML, attributes provide additional information about elements:

```
<img  src="computer.gif">
<a  href="demo.asp">
```

Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but can be important to the software that wants to manipulate the element:

```
<file  type="gif">computer.gif</file>
```

Self Assessment

Fill in the blanks:

8. XML documents form a tree structure that starts at the ..... and branches to the leaves.
9. An XML ..... is everything from (including) the element's start tag to (including) the element's end tag.
10. XML ..... provide additional information about an element.

14.3 XML Reader and XML Writer Classes

XML reader and XML writer are defining below:

14.3.1 XML Reader Classes

They represent a reader that provides fast, non-cached, forward-only access to XML data. XmlReader provides forward-only, read-only access to a stream of XML data. The current node refers to the node on which the reader is positioned. The reader is advanced using any of the read methods and properties reflect the value of the current node. XmlReader conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations and is implemented in the following classes:

Table 14.1: XML Classes

Class	Description
XmlTextReader	Fastest implementation of XmlReader. It checks for well-formed XML, but does not support data validation. This reader cannot expand general entities and does not support default attributes.
XmlvalidatingReader	Implementation of XmlReader that can validate data using DTDs or schemas. This reader can also expand general entities and supports default attributes.
XmlNodeReader	Implementation of XmlReader that reads XML data from an XmlNode.

Source: http://msdn.microsoft.com/en-us/library/system.xml.xmlreader%28v=vs.71%29.aspx

To read strongly typed data, use the XmlConvert class. For example, the following C# code reads in data and converts it from a **String** to a **Double**.

```
Double  price  =  XmlConvert.ToDouble(reader.Value);
```

XmlReader throws an XmlException on XML parse errors. After an exception is thrown the state of the reader is not predictable. For example, the reported node type may be different than the actual node type of the current node.

Notes

14.3.2 XML Writer Classes

It represents a writer that provides a fast, non-cached, forward-only means of generating streams or files containing XML data.

```
Syntax: Public MustInherit Class XmlWriter _
   Implements IDisposable
```

XmlWriter is the constructor that initializes a new instance of the XmlWriter class.

Some properties of the XMLWriter class are:

Table 14.2: XMLWriter Class Properties

Name	Description
Setting	Get the XmlWriterSettings object used to create this XmlWriter instance.
WriteState	When overridden in a derived class, gets the state of the writer.
XmlLang	When overridden in a derived class, gets the current xml:lang scope.
XmlSpace	When overridden in a derived class, gets an XmlSpace representing the current xml:space scope.

Source: http://msdn.microsoft.com/en-us/library/system.xml.xmlwriter.aspx

The XmlWriter class supports the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations.

Note Although the Microsoft .NET Framework includes the XmlTextWriter class, which is an implementation of the XmlWriter class, in the 2.0 release, it is recommended that you use the Create method to create new XmlWriter objects. The Create method allows you to specify the features to support on the created XmlWriter object, and it also allows you to take full advantage of the new features introduced in the 2.0 release.

When you use the XmlWriter methods to output XML, the elements and attributes will not be written until you call the Close method. For example, if you are using the XmlWriter to populate an XmlDocument, until you close the XmlWriter, you will not be able to observe the written elements and attributes in the target document.

The following items are things to consider when working with the XmlWriter class.

- Exceptions thrown by the XmlWriter can disclose path information that you do not want bubbled up to the application. Your applications must catch exceptions and process them appropriately.
- The XmlWriter does not validate any data that is passed to the WriteDocType or WriteRaw methods. You should not pass arbitrary data to these methods.

Example:

The following example code shows how to use the asynchronous API to generate XML.

```
async Task TestWriter(Stream stream)
{
      XmlWriterSettings settings = new XmlWriterSettings();
      settings.Async = true;
```

Notes

```
    using (XmlWriter writer = XmlWriter.Create(stream, settings)) {
          await writer.WriteStartElementAsync("pf", "root", "http://ns");
          await writer.WriteStartElementAsync(null, "sub", null);
          await writer.WriteAttributeStringAsync(null, "att", null, "val");
          await writer.WriteStringAsync("text");
          await writer.WriteEndElementAsync();
          await writer.WriteProcessingInstructionAsync("pName", "pValue");
          await writer.WriteCommentAsync("cValue");
          await writer.WriteCDataAsync("cdata value");
          await writer.WriteEndElementAsync();
         await writer.FlushAsync();
    }
}
```

14.3.3 XML–Advantages and Disadvantages

Using XML to exchange information offers many benefits. Advantages of XML include the following:

- ***Simplicity:*** Information coded in XML is easy to read and understand, plus it can be processed easily by computers.
- ***Openness:*** XML is a W3C standard, endorsed by software industry market leaders.
- ***Extensibility:*** There is no fixed set of tags.New tags can be created as they are needed.
- ***Self-description:*** XML documents can be stored without [schemas] because they contain meta data; any XML tag can possess an unlimited number of attributes such as author or version.
- ***Contains machine-readable context information:*** Tags, attributes and element structure provide context information ... opening up new possibilities for highly efficient search engines, intelligent data mining, agents, etc.
- ***Separates content from presentation:*** XML tags describe meaning not presentation.

The look and feel of an XML document can be controlled by XSL stylesheets, allowing the look of a document (or of a complete Web site) to be changed without touching the content of the document.

Multiple views or presentations of the same content are easily rendered.

- ***Supports multilingual documents and Unicode:*** This is important for the internationalization of applications.
- ***Facilitates the comparison and aggregation of data:*** The tree structure of XML documents allows documents to be compared and aggregated efficiently element by element.
- ***Can embed multiple data types:*** XML documents can contain any possible data type — from multimedia data (image, sound, video) to active components (Java applets, ActiveX).
- ***Can embed existing data:*** Mapping existing data structures like file systems or relational databases to XML is simple....
- ***Provides a "one-server view" for distributed data:*** XML documents can consist of nested elements that are distributed over multiple remote servers. XML is currently the most sophisticated format for distributed data — the World Wide Web can be seen as one huge XML database.
- ***Rapid adoption by industry:*** Software AG, IBM, Sun, Microsoft, Netscape, DataChannel, SAP .

Notes

Everything has pros and cons. Some of the problems are:

- XML syntax is redundant or large relative to binary representations of similar data.
- The redundancy may affect application efficiency through higher storage, transmission and processing costs.
- XML syntax is too verbose relative to other alternative 'text-based' data transmission formats.
- No intrinsic data type support: XML provides no specific notion of "integer", "string", "boolean", "date", and so on.
- The hierarchical model for representation is limited in comparison to the relational model or an object oriented graph.
- Expressing overlapping (non-hierarchical) node relationships requires extra effort.
- XML namespaces are problematic to use and namespace support can be difficult to correctly implement in an XML parser.
- XML is commonly depicted as "self-documenting" but this depiction ignores critical ambiguities.

Case Study

Scheduled Data Exchange

Overview

The accounting and IT teams at Altova designed and developed a system that performs automated currency exchange rate updates to the company's internal business management application. This system works behind the scenes, ensuring that all pricing information remains current for both the Euro and the US Dollar.

Altova software tools are designed to solve real-world business challenges, and therefore, projects such as this are generally done in-house, without requiring external services.

This project was assigned to an IT manager who worked with Altova tools and XML technology, as well as with built-in Microsoft® Windows functionality and the Foreign Currency Exchange Rates & Indexes Table embedded within the company's SAP Business One Enterprise Resource Planning (ERP) system.

The Challenge

Altova needed to build and deploy a simple, lightweight application whereby its back-end, SAP system would be automatically updated daily with the most current exchange rate information for US Dollars (USD) and Euros (EURO). The application was to be created and implemented by an Altova IT manager using Altova software tools. This way we would not only avoid troubling our own software developers (who were busy working on enhancements for our line of application development and data management tools) and save the cost of using third-party services, but would also demonstrate the versatility and ease of use of our tools to power users.

This project required the use of several different components, some pre-existing, and some that needed to be created by the IT manager:

Contd...

Notes

Existing Components

These components are mentioned below:

- ***XML document*** — The European Central Bank (ECB), which is responsible for regulating international currency exchange, maintains a publicly available XML document that it updates daily to provide current exchange rates. This file can be viewed at: http://www.ecb.int/stats/eurofxref/eurofxref-hist.xml
- ***Database table*** — The Foreign Currency Exchange Rates & Indexes Table within our SAP system enables the storage of current data from different currencies, based on exchange rates.
- ***Altova XMLSpy®*** — XMLSpy is the industry leading XML editor and was used in this case to generate an XML Schema based on the XML instance document mentioned above.
- ***Microsoft Scheduler*** — The Scheduled Tasks feature of the Windows operating system gives users the ability to schedule recurring processes for any installed or accessible software application.
- ***Altova MapForce® graphical user interface (GUI)*** — MapForce is an application that can be used for mapping to and from a wide variety of data formats. It provides a visual data mapping interface, which allows the user to perform complex transformations using simple drag-and-drop functionality — without writing any code.
- ***MapForce Engine & command line interface (CLI)*** — The MapForce Engine can be used within custom-built data integration applications and has the ability to execute automated mappings. The MapForce CLI launches the MapForce Engine, which in this application is called by the Scheduled Task to execute an XML to database insert.

Using these available components, the IT manager set about developing a straightforward system whereby internal pricing information would be automatically updated by referencing an external resource (the ECB XML document).

The Solution

To complete this task, the IT manager devised a method for a variety of different components to interact remotely on a regularly scheduled basis.

Created Components

These components are mentioned below:

- ***XML Schema definition (XSD)*** — XML Schema is the W3C recommended language for describing the structure of an XML document. In this case, a schema for the ECB's XML document was automatically generated using Altova XMLSpy.
- ***MapForce design (.mfd) file*** — A MapForce design file defines a data integration process that can be represented visually and is used to generate and/or execute complex data transformations.
- ***Windows Scheduled Task file (.job)*** — A Scheduled Task is a simple way to run any script, program, or document at a pre-determined time and/or frequency.

In order to update the SAP database table with the constantly changing information from the ECB XML document, a simple mapping was created using the MapForce GUI. The

Contd...

Notes

mapping would take the data from the XML file and simply transform it to the database format for use by the SAP system.

Altova MapForce utilizes an XSD to create a structural tree diagram of an XML instance for mapping purposes. The XSD is essentially a bare bones representation of the XML document structure without the associated content, and MapForce uses it as a sort of stub file or table of contents to map the elements of the XML file.

The ECB XML file is not supported by an XSD, and, in order to reap the benefits that MapForce offers, one needed to be created so that its components could be mapped to the database. XMLSpy offers the capability to infer and automatically generate an XML Schema from an XML instance document, and this was an ideal feature for this project because it only took a couple of seconds to use.

Questions

1. Study and analyse the case.
2. Write down the case facts.
3. What do you infer from it?

Source: http://www.altova.com/exchange_ratecasestudy.html

Self Assessment

True or False:

11. XML Reader Classes represent a reader that provides slow access to XML data.
12. To read strongly typed data, use the XmlConvert class.
13. XmlReader throws an ReadException on XML parse errors.
14. XML Writer Classes represents a writer that provides a cached means of generating streams or files containing XML data.
15. XMLReader and XMLWriter classes are forward only.

14.4 Summary

- Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents.
- Schema is an abstract collection of metadata, consisting of a set of schema components.
- Schema documents are organized by namespace.
- XML Schema Documents usually have the filename extension ".xsd".
- Element declarations, which define properties of elements.
- Attribute declarations define properties of attributes.
- Twenty-five derived types are defined within the specification itself.
- The purpose of a DTD is to define the legal building blocks of an XML document. It defines the document structure with a list of legal elements.
- XML documents form a tree structure that starts at "the root" and branches to "the leaves".
- XML Elements: An XML element is everything from (including) the element's start tag to (including) the element's end tag.

Notes

- XML Attributes: XML elements can have attributes, just like HTML.Attributes provide additional information about an element.
- XML Reader Classes represent a reader that provides fast, non-cached, forward-only access to XML data.
- To read strongly typed data, use the XmlConvert class.
- XmlReader throws an XmlException on XML parse errors.
- XML Writer Classes represents a writer that provides a fast, non-cached, forward-only means of generating streams or files containing XML data.

14.5 Keywords

Attribute declarations: They define properties of attributes.

DTD: It defines the legal building blocks of an XML document.

Element declarations: They define properties of elements.

Extensible Markup Language (XML): It is a markup language that defines a set of rules for encoding documents.

Schema: It is an abstract collection of metadata, consisting of a set of schema components.

XML Attributes: XML elements can have attributes, just like HTML. Attributes provide additional information about an element.

XML Elements: An XML element is everything from (including) the element's start tag to (including) the element's end tag.

XML Reader Classes: They represent a reader that provides fast, non-cached, forward-only access to XML data.

XML Writer Classes: They represents a writer that provides a fast, non-cached, forward-only means of generating streams or files containing XML data.

14.6 Review Questions

1. Define XML.
2. What is XML Schemas?
3. Write a short note on Document Type Definition (DTD).
4. List the characteristics of XML.
5. How do you represent Data through XML?
6. Differentiate between HTML and XML.
7. Explain the XML Reader Classes.
8. Why do we need the XML Writer Classes?
9. What are the advantages of XML?
10. List the disadvantages of XML.

Notes

Answers: Self Assessment

1.	XML	2.	Namespaces
3.	.xsd	4.	Element
5.	Attribute	6.	25
7.	DTD	8.	Root
9.	Element	10.	Attribute
11.	False	12.	True
13.	False	14.	False
15.	True		

14.7 Further Readings

Books

Beginning Vb.Net 2003, Willis.

Building Distributed Applications with Visual Basic.NET, Dan Fox, Sams.

Object-oriented Programming with Visual Basic.Net, Hamilton.

Programming Visual Basic .NET, J. Liberty.

VB .NET Language in a Nutshell, Steven Roman, Ron Petrusha, Paul Lomax.

Visual Basic.NET Black Book, Steven Holzner.

Online links

http://pic.dhe.ibm.com/infocenter/iseries/v6r1m0/index.jsp?topic=/rzamj/rzamjintrouses.htm

http://www.codeproject.com/Articles/20113/XML-for-Beginners

http://www.w3schools.com/xml/

http://www.exforsys.com/tutorials/xml/xml-advantages.html

LOVELY PROFESSIONAL UNIVERSITY
Jalandhar-Delhi G.T. Road (NH-1)
Phagwara, Punjab (India)-144411
For Enquiry: +91-1824-521360
Fax.: +91-1824-506111
Email: odl@lpu.co.in

978-93-90164-68-4
9 789390 164684