

SATHYABAMA UNIVERSITY

(Established under Section 3, UGC Act 1956)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



SCSX1008 C# AND .NET

SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

- Overview of .NET
 - Advantages of .NET over the other languages
 - Overview of .NET binaries
 - Intermediate Language – Metadata
- 1.5. NET Namespaces
 - Common language runtime
 - Common type system
 - Common language specification
- C# fundamentals
- 1.10C# class – object
 - string formatting
 - Types – scope – Constants
- C# iteration – Control flow
- 1.14Operators
- 1.15 Array
- 1.16String
 - Enumerations
- Structures 1.19Custom namespaces
 - Object oriented programming concepts -Class – Encapsulation
 - Inheritance
 - Polymorphic
 - Casting.

SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

Overview of .NET

What is .NET?

- ◆ .NET is a platform that provides a standardized set of services.
 - ❖ It's just like Windows, except distributed over the Internet.
 - ❖ It exports a common interface so that it's programs can be run on any system that supports .NET.

.NET Framework

- ◆ A specific software framework
 - ❖ Includes a common runtime
 - ❖ Programming model for .NET
 - ❖ Platform for running .NET managed code in a virtual machine
 - ❖ Provides a very good environment to develop networked applications and Web Services
 - ❖ Provides programming API and unified language-independent development framework

The Core of .NET Framework: FCL & CLR

- ◆ Common Language Runtime
 - ❖ Garbage collection
 - ❖ Language integration
 - ❖ Multiple versioning support
 - ❖ Integrated security
- ◆ Framework Class Library
 - ❖ Provides the core functionality:
ASP.NET, Web Services, ADO.NET, Windows Forms, IO, XML, etc.

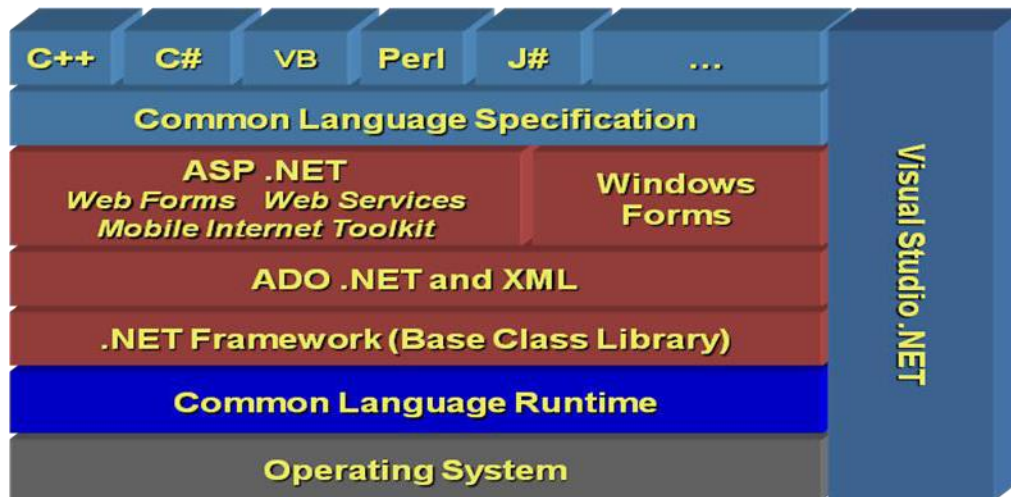
A new software platform for the desktop and the Web.

SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008



The .NET Framework is designed to fulfill the following objectives:

- To provide a consistent object-oriented programming environment where object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

ADVANTAGES OF .NET OVER THE OTHER LANGUAGES

- ➔ Language Interoperability. (means .net supports more than 40 languages writing code in one .net language can be used in other .net language)
- ➔ Language Independent platform.
- ➔ Through this technology one can develop web application as well window application (desktop application).
- ➔ Provides cross language inheritance.
- ➔ Support side by side execution.
- ➔ Memory Leak and crash Protection.
- ➔ Powerful database-driven functionality.

SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

ADVANTAGES OVER C AND C++ :

- It is compiled to an intermediate language (CIL) independently of the language it was developed or the target architecture and operating system
- Automatic garbage collection.
- Pointers no longer needed (but optional).
- Reflection capabilities .
- Don't need to worry about header files ".h"
- Definition of classes and functions can be done in any order
- Declaration of functions and classes not needed.
- There are no global functions or variables, everything belongs to a class
- All the variables are initialized to their default values before being used (this is automatic by default but can be done manually using static constructors)
- Can't use non-Boolean variables (integers, floats...) as conditions. This is much more clean and less error prone
- Apps can be executed within a restricted sandbox

ADVANTAGES OVER C++ AND JAVA

- Formalized concept of get-set methods, so the code becomes more legible
- More clean events management (using delegates)

ADVANTAGES OVER JAVA

- Usually it is much more efficient than java and runs faster
- CIL (Common (.NET) Intermediate Language) is a standard language, while java byte codes aren't.
- It has more primitive types (*value types*), including unsigned numeric types.
- Indexers let to access objects as if they were arrays.
- Conditional compilation.
- Simplified multithreading.
- Operator overloading. It can make development a bit trickier but they are optional and sometimes very useful.
- (limited) use of pointers if you really need them, as when calling unmanaged (native) libraries which doesn't run on top of the virtual machine (CLR)

Overview of .NET binaries

.NET binaries take the same file extension as classic COM binaries (*.dll or *.exe), they have absolutely no internal similarities. For example, *.dll .NET binaries do not export methods to facilitate communications with the classic COM runtime (given that .NET is not COM).

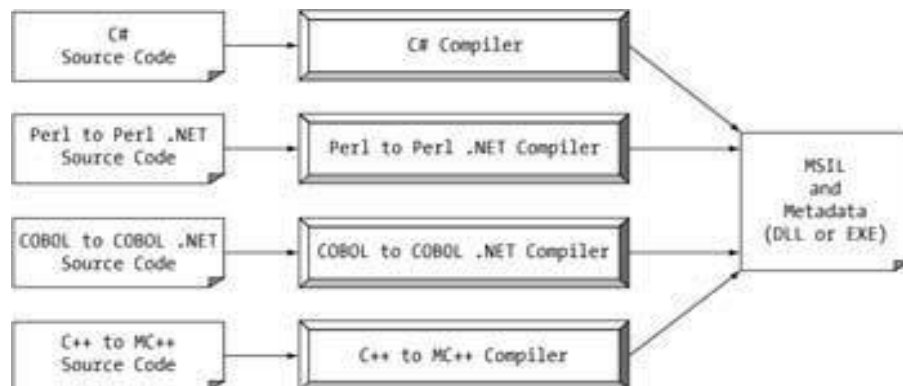
SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

Furthermore, .NET binaries are not described using IDL and are not registered into the system registry. Perhaps most important, unlike classic COM servers, .NET binaries do not contain platform-specific instructions, but rather platform-agnostic "intermediate language" (IL).



When a *.dll or *.exe has been created using a .NET-aware compiler, the resulting module is bundled into an "assembly". As mentioned, an assembly contains CIL code, which is conceptually similar to Java byte code in that it is not compiled to platform-specific instructions until absolutely necessary. Typically "absolutely necessary" is the point at which a block of CIL instructions (such as a method implementation) are referenced for use by the .NET runtime engine.

In addition to CIL instructions, assemblies also contain metadata that describes in vivid detail the characteristics of every "type" living within the binary. For example, if you have a class named Car contained within a given assembly, the type metadata describes details such as Car's base class, which interfaces are implemented by Car (if any), as well as a full description of each member supported by the Car type.

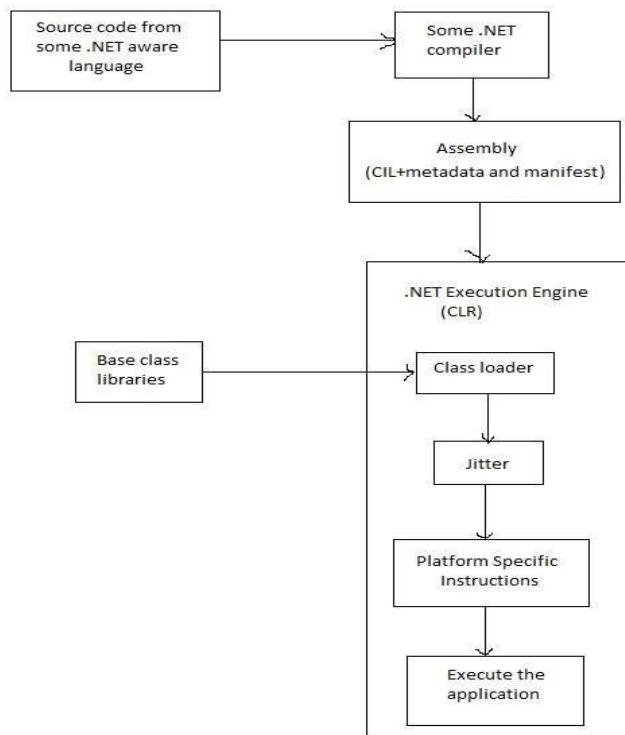
Finally, in addition to CIL and type metadata, assemblies themselves are also described using metadata, which is officially termed a *manifest*. The manifest contains information about the current version of the assembly, culture information (used for localizing string and image resources), and a list of all externally referenced assemblies that are required for proper execution.

SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008



Intermediate Language

This is the language code generated by the C# compiler or any .NET-aware compiler.

Intermediate Language: This converts native code into byte code, which is CPU –independent.
i.e. machine understandable code.

MSIL is Microsoft Intermediate Language: When we compile .Net applications, its complied to MSIL, which is not machine read language. Hence Common Language Runtime (CLR) with Just In Time Compiler (JLT) converts this MSIL to native code (binary code), which is machine language.

All .NET languages generate this code. This is the code that is executed during runtime.

This MSIL code can be viewed with the help of a utility called Intermediate Language Disassembler (ILDASM). This utility displays the application's information in a tree-like fashion. Because the contents of this file are read-only, a programmer or anybody accessing these files cannot make any modifications to the output generated by the source code.

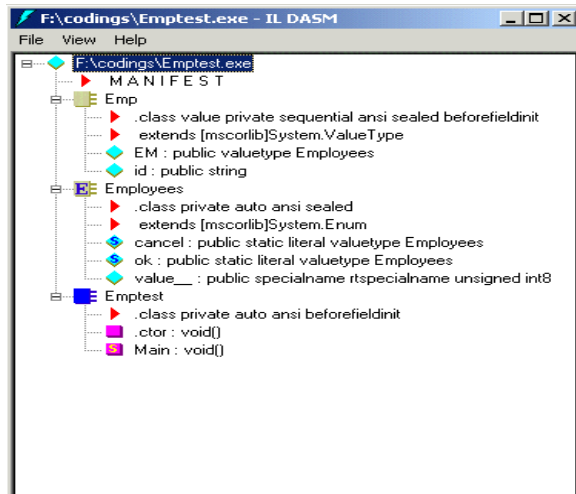
SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

To view the MSIL Code for a sample Hello C# program, open the Hello.exe file from the ILDASM tool; it can be accessed via the Run command on the Start menu. Locate this tool manually. Normally, it will be in the Bin folder of the .NET SDK installation. Figure 1 shows an outline of this tool.



The main advantages of IL are:

1. IL is not dependent on any language and there is a possibility to create applications with modules that were written using different .NET compatible languages.
2. Platform independence - IL can be compiled to different platforms or operating systems.

Meta data

Metadata is machine-readable information about a resource, or "data about data." In .NET, metadata includes type definitions, version information, external assembly references, and other standardized information.

Metadata describes contents in an assembly classes, interfaces, enums, structs, etc., and their containing namespaces, the name of each type, its visibility/scope, its base class, the interfaces it implemented, its methods and their scope, and each method's parameters, type's properties, and so on.

SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

.NET Metadata

Within the Common Language Runtime (CLR), metadata is data that describes the state of the assembly and a detailed description of each type, attribute within the assembly. Metadata includes the following information which is contained in a *manifest*. ILDASM is a .NET disassembler which allows a developer to look at the metadata (manifest) of an assembly

Assembly Metadata: Identity, types, dependent assemblies, and security permissions required for the assembly.

Type Metadata : Name, bases, interfaces, visibility and members.

Attributes : Descriptive elements that annotate assemblies, types, and members.

.NET Framework Class Library

The .NET Framework class library is a collection of reusable types that tightly integrate with the common language runtime. The class library is object oriented, providing types from which your own managed code can derive functionality. This not only makes the .NET Framework types easy to use, but also reduces the time associated with learning new features of the .NET Framework. In addition, third-party components can integrate seamlessly with classes in the .NET Framework.

For example, the .NET Framework collection classes implement a set of interfaces that you can use to develop your own collection classes. The .NET Framework types enable you to accomplish a range of common programming tasks, including tasks such as string management, data collection, database connectivity, and file access. In addition to these common tasks, the class library includes types that support a variety of specialized development scenarios. For example, you can use the .NET Framework to develop the following types of applications and services:

- Console applications.
- Windows GUI applications (Windows Forms).
- ASP.NET applications.
- XML Web services.
- Windows services.

SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

NAMESPACES

C# does not come with a pre-defined set of languages specific classes, there is no C# class library. C# uses the existing types supplied by the .NET framework. To keep all the types within this binary well organized the .NET platform makes extensive use of namespace concept. The difference between language specific library such as MFC, is that any language targeting the .NET runtime makes use of same namespace.

System is the root namespace for other .NET namespaces.

Namespaces are a way to group semantically related types.

A declarative region that provides a way to keep one set of names separate from another, i.e. the names declared in one namespace will not conflict with the same names in another namespace.

Within an Namespace we can declare

- *classes
- *structures
- *delegates
- *Enumerations
- *Interfaces
- *and another Namespaces.

Pre defined .NET NAMESPACES:-

.NET namespace	Meaning
System	Contains low level classes dealing with primitive types mathematical and manipulations
System.Collections	This namespace defines a number of stock container objects(ArrayList, Queue) as well as base types and interfaces allow you to build customized collections
System.Data	Used for database manipulations.
System.Diagnostics	Used to debug & trace your code
System.Drawing	Contains various primitives such as bitmaps, fonts, icons etc
System.IO	Includes file IO, buffering
System.NET	Contains types related to network programming
System.Security	Contains numerous types dealing with permissions, cryptography etc.
System.Threading	Deals with threading issues
System.Web	Deals with web Applications

SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

.NET namespace

Meaning

System.Windows.Forms

Contains the types that facilitate the construction of windows, dialogboxes etc.

syntax:

```
namespace name
{
....
....
....
}
```

Example:

```
using System;
class AA
{
public static void Main()
{
Console.WriteLine("we are inside namespace system");
}
}
```

output:

we are inside namespace system

COMMON LANGUAGE RUNTIME Features

of the Common Language Runtime

The common language runtime manages memory, thread execution, code execution, code safety verification, compilation, and other system services. These features are intrinsic to the managed code that runs on the common language runtime.

With regards to security, managed components are awarded varying degrees of trust, depending on a number of factors that include their origin (such as the Internet, enterprise network, or local computer). This means that a managed component might or might not be able to perform file-access operations, registry-access operations, or other sensitive functions, even if it is being used in the same active application.

The runtime enforces code access security. For example, users can trust that an executable embedded in a Web page can play an animation on screen or sing a song, but cannot access

SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

their personal data, file system, or network. The security features of the runtime thus enable legitimate Internet-deployed software to be exceptionally feature rich.

The runtime also enforces code robustness by implementing a strict type-and-code-verification infrastructure called the common type system (CTS). The CTS ensures that all managed code is self-describing. The various Microsoft and third-party language compilers generate managed code that conforms to the CTS. This means that managed code can consume other managed types and instances, while strictly enforcing type fidelity and type safety.

In addition, the managed environment of the runtime eliminates many common software issues. For example, the runtime automatically handles object layout and manages references to objects, releasing them when they are no longer being used. This automatic memory management resolves the two most common application errors, memory leaks and invalid memory references.

The runtime also accelerates developer productivity. For example, programmers can write applications in their development language of choice, yet take full advantage of the runtime, the class library, and components written in other languages by other developers. Any compiler vendor who chooses to target the runtime can do so. Language compilers that target the .NET Framework make the features of the .NET Framework available to existing code written in that language, greatly easing the migration process for existing applications.

While the runtime is designed for the software of the future, it also supports software of today and yesterday. Interoperability between managed and unmanaged code enables developers to continue to use necessary COM components and DLLs.

The runtime is designed to enhance performance. Although the common language runtime provides many standard runtime services, managed code is never interpreted. A feature called just-in-time (JIT) compiling enables all managed code to run in the native machine language of the system on which it is executing. Meanwhile, the memory manager removes the possibilities of fragmented memory and increases memory locality-of-reference to further increase performance.

Finally, the runtime can be hosted by high-performance, server-side applications, such as Microsoft® SQL Server™ and Internet Information Services (IIS). This infrastructure enables you to use managed code to write your business logic, while still enjoying the superior performance of the industry's best enterprise servers that support runtime hosting.

SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

Common Language Runtime

- ◆ Manages running code – like a virtual machine
 - ❖ Threading
 - ❖ Memory management
 - ❖ No interpreter: JIT-compiler produces native code – during the program installation or at run time
- ◆ Fine-grained evidence-based security
 - ❖ Code access security
 - ❖ Code can be verified to guarantee type safety
 - ❖ No unsafe casts, no un-initialized variables and no out-of-bounds array indexing
 - ❖ Role-based security

Managed Code

- ◆ Code that targets the CLR is referred to as managed code
- ◆ All managed code has the features of the CLR
 - ❖ Object-oriented
 - ❖ Type-safe
 - ❖ Cross-language integration
 - ❖ Cross language exception handling
 - ❖ Multiple version support
- ◆ Managed code is represented in special Intermediate Language (IL)

Automatic Memory Management

- ◆ The CLR manages memory for managed code
 - ❖ All allocations of objects and buffers made from a *Managed Heap*
 - ❖ Unused objects and buffers are cleaned up automatically through *Garbage Collection*
- ◆ Some of the worst bugs in software development are not possible with managed code
 - ❖ Leaked memory or objects
 - ❖ References to freed or non-existent objects
 - ❖ Reading of uninitialised variables
- ◆ Pointerless environment

Multiple Language Support

- ◆ IL (MSIL or CIL) – Intermediate Language
 - ❖ It is low-level (machine) language, like Assembler, but is Object-oriented
- ◆ CTS is a rich type system built into the CLR
 - ❖ Implements various types (int, float, string, ...)
 - ❖ And operations on those types
- ◆ CLS is a set of specifications that all languages and libraries need to follow
 - ❖ This will ensure interoperability between languages

SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

Intermediate Language

- ◆ .NET languages are compiled to an Intermediate Language (IL)
- ◆ IL is also known as MSIL or CIL
- ◆ CLR compiles IL in just-in-time (JIT) manner – each function is compiled just before execution
- ◆ The JIT code stays in memory for subsequent calls
- ◆ Recompilations of assemblies are also possible

COMMON TYPE SYSTEM (CTS)

The common type system defines how types are declared, used, and managed in the runtime, and is also an important part of the runtime's support for cross-language integration.

The common type system performs the following functions:

- Establishes a framework that helps enable 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.
- Encapsulates data structures.
- ◆ All .NET languages have the same primitive data types. An *int* in C# is the same as an *int* in VB.NET
- ◆ When communicating between modules written in any .NET language, the types are guaranteed to be compatible on the binary level
- ◆ Types can be:
 - ❖ Value types – passed by value, stored in the stack
 - ❖ Reference types – passed by reference, stored in the heap
- ◆ Strings are a primitive data type now.

There are two general types of categories in .Net Framework that Common Type System support. They are value types and reference types. Value types contain data and are user-defined or built-in. they are placed in a stack or in order in a structure. Reference types store a reference of the value's memory address. They are allocated in a heap structure. There are many other types that can be defined under Value types and Reference types.

SATHYABAMA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COURSE MATERIAL

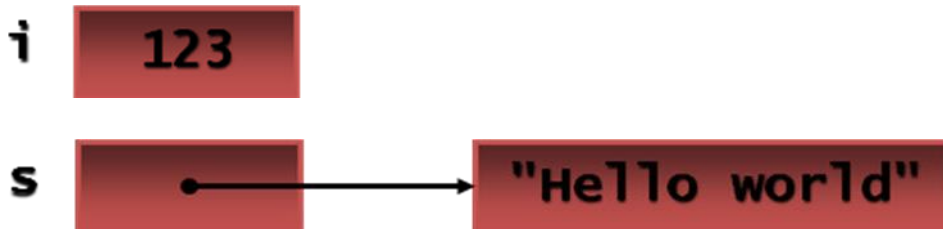
Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

Type System

- Value types
 - ☐ Directly contain data
 - ☐ Cannot be null
- Reference types
 - ☐ Contain references to objects
 - ☐ May be null
- `int i = 123;`
- `string s = "Hello world";`



- Value types
 - ☐ Primitives
 - ☐ Enums
 - ☐ Structs
 - Reference types
 - ☐ Classes
 - ☐ Interfaces
 - ☐ Arrays
 - ☐ Delegates
- ```
int i;
enum State { Off, On }
struct Point { int x, y; }

class Foo: Bar, IFoo {...}
interface IFoo: IBar {...}
string[] a = new string[10];
delegate void Empty();
```

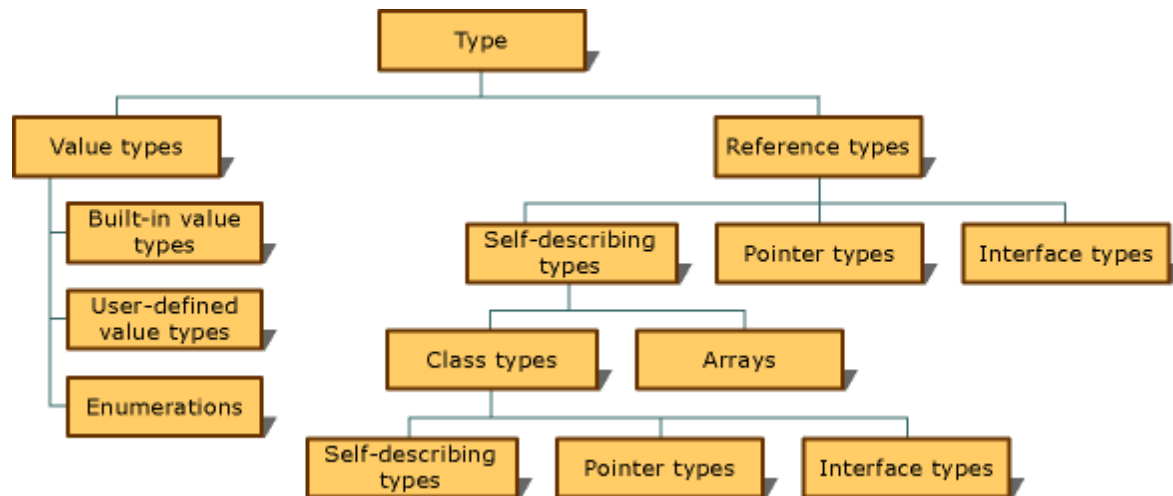
**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008

**Type classification**



**CTS Structure Types**

The concept of a structure is also formalized under the CTS. In general, a structure can be thought of as a lightweight class type having value semantics. For example, CTS structures may define any number of parameterized constructors (the no-argument constructor is reserved). In this way, you are able to establish the value of each field during the time of construction.

While structures are best suited for modeling geometric and mathematical types, the following type offers a bit more pizzazz.

**// Create a C# structure.**

```
struct Baby
{
```

**// Structures can contain fields.**

```
 public string name;
```

**// Structures can contain constructors (with arguments).**

```
 public Baby(string name)
 { this.name = name; }
```

**// Structures may take methods.**

```
 public void Cry()
 { Console.WriteLine("Waaaaaaaaaaaaah!!!"); }
 public bool IsSleeping() { return false; }
```



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

```
public bool IsChanged() { return false; }

}
```

Here is our structure in action (assume this logic is contained in some Main() method):

```
// Welcome to the world Max Barnaby!!
Baby barnaBaby = new Baby("Max");
Console.WriteLine("Changed?: {0} ", barnaBaby.IsChanged());
Console.WriteLine("Sleeping?: {0} ", barnaBaby.IsSleeping());

// Show your true colors Max...
for(int i = 0; i < 10000; i++)
 barnaBaby.Cry();
```

As you will see, all CTS structures are derived from a common base class: **System.ValueType**. This base class configures a type to behave as a stack-allocated entity rather than a heap-allocated entity. CTS permits structures to implement any number of interfaces; however, structures may not function as the base type to other classes or structures and are therefore explicitly "sealed."

### **CTS Enumeration Types**

Enumerations are a handy programming construct that allows you to group name/value pairs under a specific name. For example, assume you are creating a video game application that allows the user to select one of three player types (Wizard, Fighter, or Thief). Rather than keeping track of raw numerical values to represent each possibility, you could build a custom enumeration:

```
// A C# enumeration.
```

```
public enum PlayerType
```

```
{ Wizard = 100, Fighter = 200, Thief = 300 };
```

By default, the storage used to hold each item is a System.Int32 (i.e., a 32-bit integer), however it is possible to alter this storage slot if need be (e.g., when programming for a low memory device such as a Pocket PC). Also, the CTS demands that enumerated types derive from a common base class, System.Enum.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

### **COMMON LANGUAGE SPECIFICATION (CLS)**

To fully interact with other objects regardless of the language they were implemented in, objects must expose to callers only those features that are common to all the languages they must interoperate with. For this reason, the Common Language Specification (CLS), which is a set of basic language features needed by many applications, has been defined. The CLS rules define a subset of the Common Type System; that is, all the rules that apply to the common type system apply to the CLS, except where stricter rules are defined in the CLS. The CLS helps enhance and ensure language interoperability by defining a set of features that developer can rely on to be available in a wide variety of languages. The CLS also establishes requirements for CLS compliance; these help you determine whether your managed code conforms to the CLS and to what extent a given tool supports the development of managed code that uses CLS features.

If your component uses only CLS features in the API that it exposes to other code (including derived classes), the component is guaranteed to be accessible from any programming language that supports the CLS. Components that adhere to the CLS rules and use only the features included in the CLS are said to be CLS-compliant components.

Most of the members defined by types in the .NET Framework Class Library are CLS-compliant. However, some types in the class library have one or more members that are not CLS-compliant. These members enable support for language features that are not in the CLS. The types and members that are not CLS-compliant are identified as such in the reference documentation, and in all cases a CLS-compliant alternative is available. For more information about the types in the .NET Framework class library, see the .NET Framework Class Library.

The CLS was designed to be large enough to include the language constructs that are commonly needed by developers, yet small enough that most languages are able to support it. In addition, any language constructs that makes it impossible to rapidly verify the type safety of code was excluded from the CLS so that all CLS-compliant languages can produce verifiable code if they choose to do so. For more information about verification of type safety, see Managed Execution Process.

The Common Language Specification (CLS), which is a set of basic language features needed by many .Net applications to fully interact with other objects regardless of the language in which they were implemented. The CLS represents the guidelines defined by for the .NET Framework. These specifications are normally used by the compiler developers and are available for all languages, which target the .NET Framework.

The CLS helps enhance and ensure language interoperability by defining a set of features that developer can rely on to be available in a wide variety of languages.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

- ◆ Any language that conforms to the CLS is a .NET language
- ◆ A language that conforms to the CLS has the ability to take full advantage of the Framework Class Library (FCL)
- ◆ CLS is standardized by ECMA

**.NET Languages**

- ◆ Languages provided by Microsoft
  - ❖ C++, C#, J#, VB.NET, JScript
- ◆ Third-parties languages
  - ❖ Perl, Python, Pascal, APL, COBOL, Eiffel, Haskell, ML, Oberon, Scheme, Smalltalk...
- ◆ Advanced multi-language features
  - ❖ Cross-language inheritance and exceptions handling
- ◆ Object system is built in, not bolted on
  - ❖ No additional rules or API to learn

**C# Language Fundamentals**

- ◆ Mixture between C++, Java and Delphi
- ◆ Component-oriented
  - ❖ Properties, Methods, Events
  - ❖ Attributes, XML documentation
  - ❖ All in one place, no header files, IDL, etc.
  - ❖ Can be embedded in ASP+ pages
- ◆ Everything really is an object
  - ❖ Primitive types aren't magic
  - ❖ Unified type system == Deep simplicity
  - ❖ Improved extensibility and reusability

**C# Language – Example**

```
using System;
class HelloWorld
{
 public static void main()
 {
 Console.WriteLine("Hello, world!");
 }
}
```

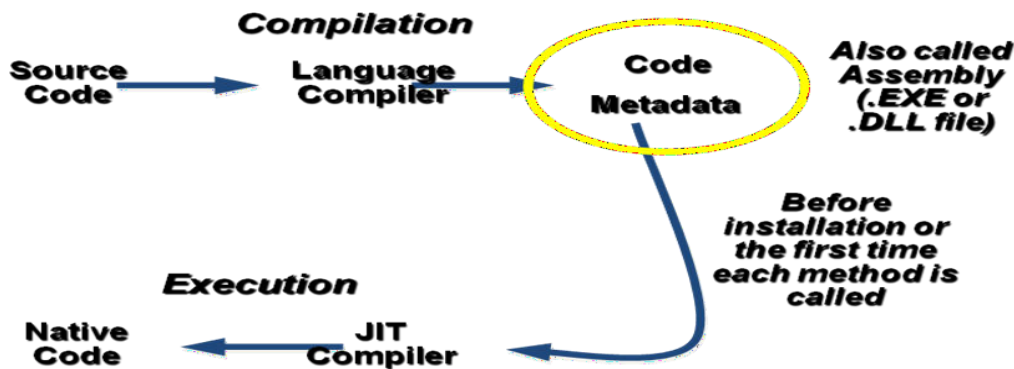
Code Compilation and Execution

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

Subject Name : C# AND .NET

UNIT I

Subject Code : SCSX1008



### ASSEMBLIES

- ◆ DLL or EXE file
- ◆ Smallest deployable unit in the CLR
- ◆ Have unique version number
- ◆ No version conflicts (known as DLL hell)
- ◆ Contains IL code to be executed
- ◆ Security boundary – permissions are granted at the assembly level
- ◆ Type boundary – all types include the assembly name they are a part of
- ◆ Self-describing manifest – metadata that describes the types in the assembly

### Classes

A *class* is a construct that enables you to create your own custom types by grouping together variables of other types, methods and events. A class is like a blueprint. It defines the data and behavior of a type. If the class is not declared as static, client code can use it by creating *objects* or *instances* which are assigned to a variable. The variable remains in memory until all references to it go out of scope. At that time, the CLR marks it as eligible for garbage collection. If the class is declared as static, then only one copy exists in memory and client code can only access it through the class itself, not an *instance variable*.

### Declaring Classes

Classes are declared by using the `class` keyword, as shown in the following example:

```
Public class Customer
{
 //Fields, properties, methods and events go here...
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

The **class** keyword is preceded by the access level. Because public is used in this case, anyone can create objects from this class. The name of the class follows the **class** keyword. The remainder of the definition is the class body, where the behavior and data are defined. Fields, properties, methods, and events on a class are collectively referred to as *class members*.

### **Creating Objects**

Although they are sometimes used interchangeably, a class and an object are different things. A class defines a type of object, but it is not an object itself. An object is a concrete entity based on a class, and is sometimes referred to as an instance of a class.

Objects can be created by using the new keyword followed by the name of the class that the object will be based on, like this:

```
Customer object1 = new Customer();
```

When an instance of a class is created, a reference to the object is passed back to the programmer. In the previous example, object1 is a reference to an object that is based on Customer. This reference refers to the new object but does not contain the object data itself. In fact, you can create an object reference without creating an object at all:

```
Customer object2;
```

This syntax is not recommended for creating object references because trying to access an object through such a reference will fail at run time. However, such a reference can be made to refer to an object, either by creating a new object, or by assigning it to an existing object, such as this:

```
Customer object3 = new Customer();
Customer object4 = object3;
```

This code creates two object references that both refer to the same object. Therefore, any changes to the object made through object3 will be reflected in subsequent uses of object4. Because objects that are based on classes are referred to by reference, classes are known as reference types.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

**String Formatting**

**String.Format Method**

Converts the value of objects to strings based on the formats specified and inserts them into another string.

**Formatting Console Output**

.NET introduces a new style of string formatting, slightly reminiscent of the C printf() function, but without the cryptic %d, %s, or %c flags. A simple example follows

```
static void Main(string[] args)
{
 ...
 int theInt = 90;
 double theDouble = 9.99;
 bool theBool = true;
 // The '\n' token in a string literal inserts a newline.
 Console.WriteLine("Int is: {0}\nDouble is: {1}\nBool is: {2}",
 theInt, theDouble, theBool);
}
```

The first parameter to WriteLine() represents a string literal that contains optional placeholders designated by {0}, {1}, {2}, and so forth (curly bracket numbering always begins with zero). The remaining parameters to WriteLine() are simply the values to be inserted into the respective placeholders (in this case, an int, a double, and a bool).

:

**// Fill placeholders using an array of objects.**

```
object[] stuff = {"Hello", 20.9, 1, "There", "83", 99.99933};
Console.WriteLine("The Stuff: {0} , {1} , {2} , {3} , {4} , {5} ", stuff);
```

It is also permissible for a given placeholder to repeat within a given string. For example, if you are a Beatles fan and want to build the string "9, Number 9, Number 9" you would write

```
Console.WriteLine("{0}, Number {0}, Number {0}", 9);
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

**SCOPE**

**Definition**

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.

**Scoping 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 accessed by code in the same project/assembly.                      |
| ProtectedFriend | Variables/Properties/Methods       | Combination of Protected and Friend                                             |

---

**Constants**

- The variables whose values do not change during the execution of a program are known as constants.
- Const keyword is used to declare constants.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

Eg: Const int n = 10;

Note: Names of constants take the same form as variables names

- After declaration of constants they should not be assigned any other value
- Constants can not be declared inside a method. They should be declared only at class level

## **Control Structures**

### **ITERATION STATEMENTS**

Iterative statements repeat a particular statement block until a condition has been satisfied.

Following types of iterative statements are there in C#.

1. While
2. Do While
3. For
4. For each

### **While Statement**

The statement block of while statement is executed while the boolean expression is true. It may execute zero or more times. If the boolean expression is false initially, the statement block is not executed.

```
int i = 9;
int j = 7;
int sum = 0;
while (j < i)
{
 sum += j;
 j++;
}
```

The value in the *sum* variable will be 15 as this loop will continue 2 times only. As soon as the value of *j* will reach to 9 the condition *j < i* will return false and the loop will break.



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

The while statement is used to iterate while the condition evaluates to true.

### **Do While Statement**

Do While is almost similar to While statement except it validate its Boolean expression after the statement block. It guarantees that the statement block shall run at least once for sure. Further iterations of the statement block continues while the Boolean expression is true.

```
int i = 9;
int j = 10;
int sum = 0;
do
{
 sum += j;
 j++;
} while (j < i);
```

The value in the *sum* variable will be 10 as *j* is already greater than *i* so after first time entering into the loop, the boolean validation (*j < i*) will return false and loop will break. The above code block may not be the good example of the do while loop; do while can be used to ask for the desired input from the user. If we are not getting the desired input, we can continue asking the question in the while statement block.

### **For Statement**

The For statement iterate a code block until a specified condition is reached similar to while statement. The only difference of for statement has over while statement is that for statement has its own built-in syntax for initializing, testing, and incrementing/decrementing (3 clauses) the value of a counter.

*The first clause* is the initialize clause in which the loop iterators are declared.

*The second clause* is the boolean expression that must evaluate to a boolean type and the statement block is repeated until this expression is false.

*The third clause* is to increment/decrement the value that is executed after each iteration.

All three clauses must be separated by a semicolon (;) and are optional. For statement block is repeated zero or more times based on the boolean expression validation (second clause).

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

```
int j = 10;
int sum = 0;
for (int i = 0; i < j; i++)
{
 sum += i;
}
```

The value of *sum* variable will be 45 (0+1+2+3+4+5+6+7+8+9). It will add all the number from 0 to 9 because once it will reach 10 (third clause increase the value of *i* after each iteration), the second clause (boolean expression) will not satisfy and loop will break.

### **For Each Statement**

For Each statement is designed to loop through a similar type of items in a collection. As each element is enumerated, the identifier is assigned the new element, and the statement block is repeated. The scope of the identifier is within the for each statement block. The identifier must be of the same type extracted from the collection and is read-only.

```
string[] days = { "sunday", "monday", "tuesday" };
string output = string.Empty;
foreach (string s in days)
{
 output = string.Concat(output, s + " > ");
}
```

The identifier of the above *foreach* loop is *string s* (because *days* array contains string so identifier must be of the same type). The value of the *output* variable will be "sunday > monday > tuesday > ". In the *days* array, we have three strings and in the for each loop we are concatenating all the items of the loop.

### ***Use of break and continue***

If we want to break the loop for some reason (may be after looping and validating the item, we got the desired result); we can use break (break;) statement to prematurely exist from the loop.

If we want to skip a particular iterations and continue next iteration; we can use continue (continue;) statement. Continue statement transfers the control to the end of the statement block where the next execution continues.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

Subject Name : C# AND .NET

UNIT I

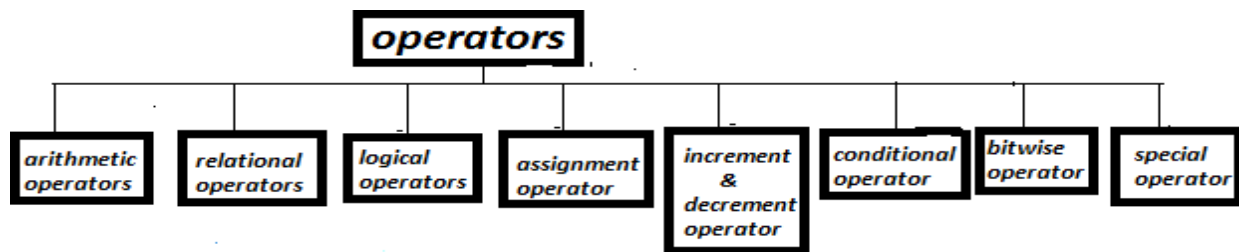
Subject Code : SCSX1008

## OPERATORS

---

An operator is a symbol that tells the computer to perform mathematical or logical manipulations.

Operators are used in programs to manipulate data and programs.



### *Arithmetic operators:*

| <i>Operators</i> | <i>meaning</i>                    |
|------------------|-----------------------------------|
| <i>+</i>         | <i>addition or unary plus</i>     |
| <i>-</i>         | <i>subtraction or unary minus</i> |
| <i>*</i>         | <i>multiplication</i>             |
| <i>/</i>         | <i>division</i>                   |
| <i>%</i>         | <i>modulo division</i>            |

- The Operators +, -, \* and / all work the same way as they do in other languages.
- These can operate any built-in numeric data type. We cannot use these operators in Boolean type.

Arithmetic Operators are used as:

|     |      |
|-----|------|
| a-b | a+b  |
| a*b | a/b  |
| a%b | -a*b |

Here a and b may be variables or constants and are known as Operands.

### **Integer Arithmetic**

- When both the operands in a single arithmetic expression such as a+b are integers, the expression is called an integer expression, and the operation is called integer arithmetic.
- Integer arithmetic always yields an integer value.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

Example:

a=14 and b=4

a-b=10

a+b=18

a\*b=56

a/b=3

**Real Arithmetic**

- An arithmetic Operation involving only real operands is called real arithmetic.
- A real operand may assume values either in decimal or exponential notation.
- Modulus operand returns the floating-point equivalent of an integer division.

Example:

a=20.5

b=6.4

a+b=26.9

a-b=14.1

a\*b=131.2

a/b=3.203125

**Mixed-mode Arithmetic**

- When one of the operands is real and other is integer, the expression is called a mixed-mode arithmetic expression.

Example:

15/10.0 produces the result 1.5

15/10 produces the result 1

**Relational Operators**

- We often compare two quantities and depending on their relation, take certain decisions. For example, we may compare the age of two persons or the price of two items and so on. These Comparisons can be done with the help of relational operators.

Example:

a<b or x<20

Expression containing a relational operator is termed as a relational expression. The value of a relational expression is either true or false. For example, Relational expression are

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

used in decision statements such as if and while to decide the course of action of a running program.

| <i>Operators</i> | <i>meaning</i>                     |
|------------------|------------------------------------|
| <                | <i>is less than</i>                |
| <=               | <i>is less than or equal to</i>    |
| >                | <i>is greater than</i>             |
| >=               | <i>is greater than or equal to</i> |
| ==               | <i>is equal to</i>                 |
| !=               | <i>is not equal to</i>             |

**Logical Operators**

- The logical operators && and || are used when we want to form compound condition by combining two or more relations. An example is:  
a>b && x==10
- An expression of this kind which combines two or more relational expression is termed as a logical expression or a compound expression. Like the simple relational expressions, a logical expression also yields a value of true or false.

| <i>operators</i> | <i>meaning</i>                      |
|------------------|-------------------------------------|
| &&               | <i>logical AND</i>                  |
|                  | <i>logical OR</i>                   |
| !                | <i>logical NOT</i>                  |
| &                | <i>bitwise logical AND</i>          |
|                  | <i>bitwise logical OR</i>           |
| ^                | <i>bitwise logical exclusive OR</i> |

**Assignment Operators**

- Assignment operators are used to assign the value of an expression to a variable.
- Assignment operator, '='.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

Example:

$v \text{ OP} = \text{exp}$

Where  $v$  is a variable,  $\text{exp}$  is an expression and  $\text{op}$  is a C# binary operator.

- The operator  $\text{OP} =$  is known as the shorthand assignment operator.
- The shorthand operator  $y += x$  means 'add  $y+1$  to  $x$ ' or 'increment  $x$  by  $y+1$ '.  $X += 3$ ;
- The use of shorthand assignment operators has three advantages

\*What appears on the left-hand side need not be repeated and therefore it becomes easier to write

\*The statement is more concise and easier to read

\*The use of shorthand operators results in a more efficient code.

| <i>Statement with simple assignment operators</i> | <i>statements with shorthand operator</i> |
|---------------------------------------------------|-------------------------------------------|
| $a = a + 1$                                       | $a += 1$                                  |
| $a = a - 1$                                       | $a -= 1$                                  |
| $a = a * (n + 1)$                                 | $a *= n + 1$                              |
| $a = a / (n + 1)$                                 | $a /= n + 1$                              |
| $a = a \% b$                                      | $a \% = b$                                |

### **Increment and Decrement Operators**

- There are the increment and decrement operators:

$++$  and  $--$

- The operator  $++$  adds 1 to the operand while  $--$  subtracts 1.
- Both are unary operators and are used in the following form:

$++m$ ; or  $m++$ ;

$--m$ ; or  $m--$ ;

$++m$ ; is equivalent to  $m = m + 1$ ; (or  $m + = 1$ ;)

$--m$ ; is equivalent to  $m = m - 1$ ; (or  $m - = 1$ ;)

- We use the increment and decrement operators extensively in for and while loops,
- While  $++m$  and  $m++$  mean the same thing when they form statements independently, they behave differently when they are used in expressions on the right-hand side of an assignment statement, Consider the following:

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

```
m=5;
y= ++m;
```

**Example:**

```
m=10
n=20
++m=11
n+=20
m=11
n=21
```

**Conditional Operator**

- The character pair ?: is a temporary operator available in C#. This operator is used to construct conditional expressions of the form

exp1?exp2:exp3

where exp1,exp2,exp3 are the expressions.

- The operator?: works as follows: exp1 is evaluated first. If it is true, then the expression exp2 is evaluated and becomes the value of the conditional expression. if exp 1 is false,exp3 is evaluated and its value becomes the value of the conditional expression. Note that only one of the expressions (either exp2 or exp3) is evaluated .For example, consider the following statements:

```
a=10;
b=15;
x=(a>b) ? a:b; equivalent to,
if (a > b)
x=a;
else
x=b;
```

**Bitwise Operators**

C# supports the operators that may be used for manipulation of the data at bit level. These operators may be used for testing the bits or shifting them to the right or left. Bitwise operators may not be applied to floating point data.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

| <i>operator</i> | <i>meaning</i>             |
|-----------------|----------------------------|
| <b>&amp;</b>    | <i>bitwise logical AND</i> |
| <b> </b>        | <i>bitwise logical OR</i>  |
| <b>^</b>        | <i>bitwise logical XOR</i> |
| <b>~</b>        | <i>one's complement</i>    |
| <b>&lt;&lt;</b> | <i>shift left</i>          |
| <b>&gt;&gt;</b> | <i>shift right</i>         |

### **Special operators**

C# supports the following special operators:

Is (relational operator)  
as (relational operator)  
typeof (type operator)  
sizeof (size operator)  
new (object creator)  
.(dot) (member-access operator)  
Checked (overflow checking)  
Unchecked (prevention of overflow checking)

### **Array**

Array is the group of related data items. Length property is used to find and the length of the array.

#### **One dimensional Array:**

Syntax: datatype[] arrayname = new datatype[size];  
eg: int[] a = new int[10];

Initialization of One Dimensional Array:

int[] a={ 10, 20, 30};

Accessing array elements:

a[1] => 1<sup>st</sup> element of array a

Eg:Program to find sum of Array elements

using System;



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

```
class DemoArray
{
 public static void Main()
 {
 int[] a={ 10,20,30};
 int s=0;
 for(int i=0; i<3; i++)
 s=s+a[i];
 Console.WriteLine("Sum is "+s);
 }
}
```

**Output :** Sum is 60

**Multi Dimensional Array:**

Contains many rows.

Syntax:

datatype[,] arrayname=new datatype [row size][column size];

eg:

int [,] a=new int [2][3];

Accessing array elements:

a[2,3]=> 2<sup>nd</sup> row, 3<sup>rd</sup> element

**Eg:Program to print elements of two dimensional array**

```
class TDArrary
{
 public static void Main()
 {
 int[,] a={{ 10,20},{ 30,40 } };
 for (int i=0; i<a.GetLength(0); i++)
 {
 for (int j=0; j<a.GetLength(1); j++)
 {
 Console.Write(a[i,j]);
 }
 Console.WriteLine("\n");
 }
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

**Output :**

10 20  
30 40

**Jagged Array:**

Jagged Array contains some numbers of inner arrays, each of which may have a Unique Upper Limit.

Syntax:

```
int[][] JA=new int[2][];
```

Ex:

```
JA[0]=new int[2];
```

```
JA[1]=new int[3];
```

Now the jagged array contains 2 rows.

1<sup>st</sup> row contains 2 elements,

2<sup>nd</sup> row contains 3 elements.

**Length Property:**

For one and multi dimensional arrays, the length property returns no. of elements in that array.

Length can be used for Jagged Arrays. Here it is possible to obtain the length of each individual array.

Eg:

```
JA.Length = 2[no. of rows]
```

```
JA[1].Length = 2[no. of elements in 1st row]
```

Note:

By default the values of array elements are zero.

**Methods of class System.Array:**

C# array is derived from System.Array (.Net Base Type). Hence C# arrays can use the members of System.Array.

Binary Search()

Reverse()

Clear()

Sort()

eg: int a={40, 20, 30, 10}

```
int i=Array.BinarySearch(a,20)
```

Now i=1

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

```
Array.Sort(a)
 Now a= { 10, 20, 30, 40}
Array.Reverse(a)
 Now a={40, 30, 20, 10}
Array.Clear(a, 0, 2)
 Now a={0, 0, 20, 10}
```

### **String**

One of the data type is string. Used to store strings. Strings are objects. Hence it is a reference type.

#### **Declaration of strings:**

```
string s1= "Hello";
```

String is derived from the base class System.String (.Net base type). By means of index, you can access individual character but can't modify it.

[Eg: s1[0]='H']

#### **Methods of class System.String:**

(1) public static string Copy(string s1)

Used to copy the string.

Eg: string s1="Hello"

```
string s2=String.Copy(s1);
s2="Hello".
```

(2) public static int Compare(string s1, string s2, bool ignorecase)

Used to compare strings.

ignorecase --> true => ignore case

ignorecase --> false => won't ignore case

eg: string s1="abc"

```
string s2="ABC"
```

```
int i=String.Compare(s1,s2,false)
```

```
i=65-97= -32 [s1<s2]
```

(3) public static string concat (string s1, string s2)

Used to concatenate two strings.

Eg: string s1 = "Good"

```
string s2 = "Morning"
```

```
string s3 = String.concat(s1,s2);
```

```
Now string3 = Good Morning
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

(4) public int IndexOf(string str or char c)

=> Searches the invoking string for the substring or character and returns the index of the first match.

```
Eg: string s1="Good";
int i = s1.IndexOf('o');
Now i=1
```

(5) public int LastIndexOf(string str) or (char c)

=> Searches the invoking string for the substring or character and returns the index of the last match.

```
Eg: string s1 = "Good"
int i = s1.LastIndexOf('o');
Now i = 2
```

(6) public bool StartsWith(String str)

Used to check whether the string starts with given substring. It returns true or false.

```
Eg: string s1="Good"
bool b = s1.StartsWith("Good");
b=true
```

(7) public bool EndsWith(string str)

Used to check whether string ends with the given substring.

```
Eg: string s1="Good Morning";
bool b=s1.EndsWith("ING");
b=false
```

(8) public string ToLower() & ToUpper()

Used to convert characters from uppercase to lowercase

```
eg: String S1 = "Good";
String S2 = S1.ToLower();
S2 = good
String S3 = S2.ToUpper();
S3 = GOOD
```

(9) Public string Replace(character to be replaces, character for replacement)

Used to replace one character with other.

```
Eg: String S1 = "Geed";
String S2 = S1.replace('e', 'o');
S2 = Good
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

(10) public string insert(string str, int location)

Used to insert the string at the specified location

Eg:

```
String S1 = This Good
 12345
String S2 = S1.insert("is", 5);
S2 = This is good.
```

(11) public string remove(int start, int count)

Used to remove the count no of character from the given string starting from the location specified by start.

Eg: String S1 = "This is good";  
String S2 = S1.remove(5,2);  
S2 = This good

## **ENUMERATIONS**

An ENUMERATION provides an efficient way to define a set of named integral constants that may be assigned to a variable.

- The Enumeration list is a comma separated list of identifiers.
- By default, the value of the first enumeration symbol is 0.

### **syntax:**

```
enum name [enumeration list];
```

### **Exmample**

```
using System;
class Program
{
 enum Importance
 {
 None,
 Trivial,
 Regular,
 Important,
 Critical
 };
};
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

```
static void Main()
{
 Importance value = Importance.Critical;
 if (value == Importance.Trivial)
 {
 Console.WriteLine("Not true");
 }
 else if (value == Importance.Critical)
 {
 Console.WriteLine("True");
 }
}
```

***Output :*** True

## **STRUCTURE**

### **Definition**

Structure is a value data type . It is used to store mixed data types. It provides a unique way of packing different data types together. It is convenient tool for handling a group of logically related data items.

### ***Syntax :***

```
struct struct-name
{
 data type member1;
 data type member2;
}

eg:
 struct Student
 {
 public string name;
 public int rollnumber;
 }
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

Program for displaying student detail using structure

```
using System;
public struct Student
{
 public string name;
 public int rollnumber;
}

class Csstruct
{
 public static void Main ()
 {
 Student S1;
 S1.name = "lakshmi";
 S1.rollnumber = 101;
 Console.WriteLine(S1.name+"\t"+S1.rollnumber);
 }
}
```

|                             |
|-----------------------------|
| <b>Output : lakshmi 101</b> |
|-----------------------------|

### **Features of C# structure**

Structures support defined constructors, but not destructors. The default constructor is automatically defined and cannot be changed.

Structures can have methods.

Structure Variable can be passed as a parameter to any member function.

### **Class versus Structure**

classes are reference types and structs are value types

structures do not support inheritance

structures cannot have default constructor

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

Program for displaying student details by defining constructors

eg:

```
struct Student
{
 public string name;
 public int rollno;
}
class CsStruct
{
 public static void Main ()
 {
 Student S1= new student("lakshmi", 101);
 Console.WriteLine(S1.name+"\t"+S1.rollno);
 }
}
```

|                             |
|-----------------------------|
| <b>Output : lakshmi 101</b> |
|-----------------------------|

Program for displaying student details by defining methods inside the structure.

eg:

```
struct student //structure
{
 public string name;
 public void Display(string n, int r) // method
 {
 name =n;
 rollno =r;
 }
}
class csstruct
{
 public static void Main()
 {
 student s1 = new student(); //structure object
 s1. Display ("lakshmi",101)
 Console.WriteLine(s1.name);
 Console.WriteLine(s1.rollno);
 }
}
```

|                             |
|-----------------------------|
| <b>Output : lakshmi 101</b> |
|-----------------------------|



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

Program for displaying student details by passing structure as parameters to any member function.

Eg:

```
struct student
{
 string name;
 int rollno;
}
class csstruct
{
 public static Void display(student s1)
 //sturcture cannot be inherited they can be passed as parameter
 {
 Console.WriteLine(s1.name+"\t"+s1.rollno);
 }
 public static Void Main()
 {
 student s1;
 s1.name = "lakshmi";
 s1.rollno = 101;
 display(s1);
 }
}
```

|                                  |
|----------------------------------|
| <b>Output : lakshmi      101</b> |
|----------------------------------|

Syntax for Copying one structure to another:

eg: student s1,s2;  
s2=s1;//now s1 is copied to s2.

Syntax for using Struct variable as a member for another structure

eg:

```
struct M
{
 public int x;
}
struct N
{
 public M m;
 public int Y;
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

### **DEFINING CUSTOM NAMESPACES**

We can also define our own namespaces using the keyword **namespace**.

**Syntax :**

```
namespace namespace_name
{
 public class calss1{ }

 public class class2{ }
}
```

Example program for defining custom namespace **named Myshapes**

```
using System;
namespace Myshapes //Myshapes is the user defined namespace
{
 public class circle
 {
 public void display()
 {
 Console.WriteLine("You have called display method of circle ");
 }
 }
 public class square
 {
 public void display()
 {
 Console.WriteLine("You have called display method of square ");
 }
 }
}
```

Program for accessing custom namespace **Myshapes**

eg:

```
using System;
using Myshapes; //import the desired namespace
class ex
{
 public static void Main()
 {
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

```
 circle c = new circle(); //circle object of the desired class
 square s = new square();
 c.display();
 s.display();
 }
}
```

|                                                                                                              |
|--------------------------------------------------------------------------------------------------------------|
| <p><b>Output</b> : You have called display method of circle<br/>You have called display method of square</p> |
|--------------------------------------------------------------------------------------------------------------|

### Resolving Name classes across Namespaces

Fully qualified name resolves the name clash across multiple namespaces

eg:

One more Namespace My3DShapes that also contain the same class circle.

```
using System;
Namespace My3Dshapes
{
 public class circle
 {}
 public class Square
 {}
}
```

Now we can differentiate circle class of MyShapes and My3DShapes by means of fully qualified names.

Eg:

```
using MyShapes;
using My3DShapes;
class ex
{
 PSVM()
 {
 MyShapes.Circle c1 = new MyShapes.circle();
 My3DShapes.Circle c2 = new My3DShapes.Circle();
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

### **Nested namespace**

Namespaces can also be nested within other namespaces

Syntax

```
using System;
namespace A
{
 namespace B
 {
 define classes //here B within A known as nested Namespaces
 }
}
```

### **Object Oriented Programming with C#**

There are three major object oriented concepts

*Encapsulation*

*Inheritance*

*Polymorphism*

### **Encapsulation**

This is the languages ability to hide unnecessary implementation details from the end user. Binding of data and functions into the single unit is also known as Encapsulation.

public data members of the class can be accessed by means of object.

Eg:

```
class F
{
 public int a = 20;
}
class F1
{
 public static void Main()
 {
 F b = new F();
 Console.WriteLine("Value of a is ", b.a);
 }
}
```

|                                  |
|----------------------------------|
| <b>Output : Value of a is 20</b> |
|----------------------------------|

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

**There are two ways to access private data members**

By means of defining two public methods

By means of defining properties

**Accessing private data by means of defining public methods**

Eg:

```
using System;
class A
{
 Private int a;
 public void set1(int x)
 {
 a = x;
 }
 public int get1()
 {
 return(a);
 }
}
class B
{
 public static void Main()
 {
 A a1 = new A();
 a1.set1(10); //access private data by means of public method.
 Console.WriteLine("Value of private data a is " + (a1.get1()));
 }
}
```

|                                               |
|-----------------------------------------------|
| <b>Output : Value of private data a is 10</b> |
|-----------------------------------------------|

**Accessing private data by means of defining properties**

This is also used to access private data. Instead of using two public methods for getting and setting values for private data, we can use single property.

Syntax:

```
type name
{
 get {}
 set {}
}
Type --> type of the property
Name --> name of the property
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

Once the property has been defined, any use of name results in a call to its appropriate accessor. The set accessor automatically receives a parameter called value that contains the value being assigned to the property.

To manipulate internal data using single variable

eg:

```
using System;
class A
{
 Private int a;
 no modifier int prop //syntax for property
 {
 get
 {
 return(a);
 }
 set
 {
 a = value;
 }
 } //property
} //class

class B
{
 public static void Main()
 {
 A a1 = new A();
 a1.prop(10); //call set accessor of property prop and set value of private data
 Console.WriteLine("Value is "+a1.Prop) //call get accessor of property prop
 }
}
```

**Three Types of properties**

Read Only Property

Write Only Property

Static Property

If the property contains only get block then that property is known as read only property.

Eg:

```
int Prop
{
 get{ }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

If the property contains only set block then that property is known as write only property

```
Eg:
int Prop
{
 Set
 {
 Private data = value;
 }
}
```

**Static property**

If the property is defined as static then that can be accessed by means of class name.

```
Eg:
class A
{
 Private static int a;
 public static int prop
 {
 get
 {
 return(a);
 }
 set
 {
 a = value;
 }
 }
}

class B
{
 public static void Main()
 {
 A.prop =10; // set accessor of static property prop is called with class name .
 Console.WriteLine("Value is "+A.prop); // call get accessor of static property prop
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

## **Inheritance**

This concept is mainly used for code reuse . Inheritance comes in two flavours

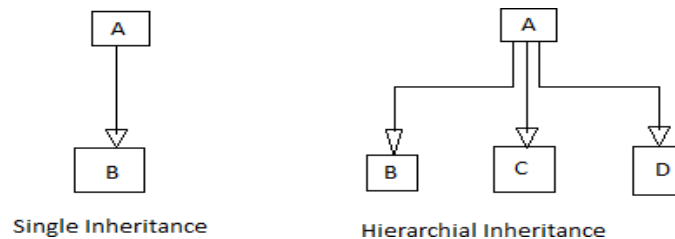
*is-a relationship (classical inheritance)*

*has-a relationship(containment)*

### **Classical inheritance**

Definition:- the basic idea behind is that new classes may leverage the functionality of other classes.

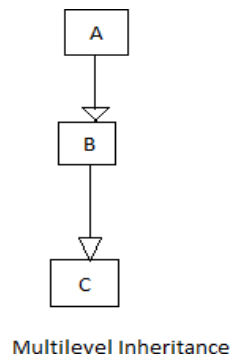
Eg:



class A --> baseclass, parent class, super class

class B --> derived class, child class, sub class

Now the derived class that incorporate all the data and methods of its base class, have its own data member. Now B is a type of A .Classical inheritance can be implemented in different combinations.





**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

**Containment inheritance:**

Containership between class A & B

Ex:

```
class A
{
}
class B
{
 A a; //a is contained in B
}
```

the relationship between A and B is has a relationship.

**General form of defining sub-class:-**

Syntax:

```
class derived class name:base class name
{
 variable declaration;
 method declaration;
}
```

ex:

```
class A
{
}
class B:A
{
}
```

**Example program for single inheritance**

```
class A
{
 public void display1()
 {
 Console.WriteLine("Base class");
 }
}
class B:A
{
 public void display2()
 {
 Console.WriteLine("Derived class");
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

```
class C
{
 public static void Main()
 {
 B b1 = new B();
 b1.display1();
 b1.display2();
 }
}
```

**OUTPUT :**

Base Class

Derived class

**Base Keyword**

Used to call base class constructor from derived class

Used to access base class data member from derived class

Example program to call base class constructor from derived class

```
class A
{
 public int i;
 public A(int x)
 {
 i = x;
 }
}
class B:A
{
 public int i1;
 public B(int x, int y):base(x)//call the base class constructor
 {
 i1 =y;
 }
}
class C
{
 public static void Main()
 {
 B b1 = new B(10,20);
 Console.WriteLine("Value of i is “ +b1.i);
 Console.WriteLine("Value of i1 is “+b1.i1);
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

**OUTPUT :**

Value of i is 10  
Value of i1 is 20

Note: if the base class and derived class has the same member, then the base class member can be accessed from the derived class using the keyword base.

Example program to access base class data member from derived class

```
class A
{
 public int i = 10;
}
class B:A
{
 public int i = 20;
 public void display()
 {
 Console.WriteLine("Value of base class i "+base.i);
 Console.WriteLine("Value of derived class i"+i);
 }
}
class c
{
 public static void Main()
 {
 B b1 = new B();
 b1.display();
 }
}
```

**OUTPUT :**

Value of base class i 10  
Value of derived class i 20

**Prevent inheritance using keyword sealed:-**

```
sealed class A
{}
class B:A//error because A cannot be inherited
{}
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

## **Polymorphism**

---

When a message can be processed in different ways is called polymorphism. Polymorphism means many forms.

Polymorphism is one of the fundamental concepts of OOP.

**Polymorphism provides following features:**

- It allows you to invoke methods of derived class through base class reference during runtime.
- It has the ability for classes to provide different implementations of methods that are called through the same name.

**Polymorphism is of two types:**

1. Compile time polymorphism/Overloading
2. Runtime polymorphism/Overriding

### **Compile Time Polymorphism**

Compile time polymorphism is method and operators overloading. It is also called early binding.

In method overloading method performs the different task at the different input parameters.

### **Runtime Time Polymorphism**

Runtime time polymorphism is done using inheritance and virtual functions. Method overriding is called runtime polymorphism. It is also called late binding.

When **overriding** a method, you change the behavior of the method for the derived class. **Overloading** a method simply involves having another method with the same prototype.

**Following are examples of methods having different prototypes:**

```
void area(int side);
void area(int l, int b);
void area(float radius);
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

**Program for Method Overloading (Compile Time Polymorphism)**

using System;

```
class Program
{
 public void display(string name)
 {
 Console.WriteLine("Your name is : " + name);
 }

 public void display(int age, float marks)
 {
 Console.WriteLine("Your age is : " + age);
 Console.WriteLine("Your marks are : " + marks);
 }
}

static void Main(string[] args)
{
 Program obj = new Program();
 obj.display("George");
 obj.display(34, 76.50f);
 Console.ReadLine();
}
}
```

**Method Overriding(Run time Polymorphism)**

When a derived class inherits from a base class, it gains all the methods, fields, properties and events of the base class. To change the data and behavior of a base class, you have two choices: you can replace the base member with a new derived member, or you can override a virtual base member.

Replacing a member of a base class with a new derived member requires the **new** keyword. If a base class defines a method, field, or property, the **new** keyword is used to create a new

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

definition of that method, field, or property on a derived class. The **new** keyword is placed before the return type of a class member that is being replaced. For example:

```
public class BaseClass
{
 public void DoWork() { }
 public int WorkField;
 public int WorkProperty
 {
 get { return 0; }
 }
}

public class DerivedClass : BaseClass
{
 public new void DoWork() { }
 public new int WorkField;
 public new int WorkProperty
 {
 get { return 0; }
 }
}
```

When the **new** keyword is used, the new class members are called instead of the base class members that have been replaced. Those base class members are called hidden members. Hidden class members can still be called if an instance of the derived class is cast to an instance of the base class. For example:

```
DerivedClass B = new DerivedClass();
B.DoWork(); // Calls the new method.
```

```
BaseClass A = (BaseClass)B;
A.DoWork(); // Calls the old method.
```

In order for an instance of a derived class to completely take over a class member from a base class, the base class has to declare that member as virtual. This is accomplished by adding the virtual keyword before the return type of the member. A derived class then has the option of using the override keyword, instead of **new**, to replace the base class implementation with its own. For example:

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

```
public class BaseClass
{
 public virtual void DoWork() { }
 public virtual int WorkProperty
 {
 get { return 0; }
 }
}
public class DerivedClass : BaseClass
{
 public override void DoWork() { }
 public override int WorkProperty
 {
 get { return 0; }
 }
}
```

Fields cannot be virtual; only methods, properties, events and indexers can be virtual. When a derived class overrides a virtual member, that member is called even when an instance of that class is being accessed as an instance of the base class. For example:

```
DerivedClass B = new DerivedClass();
B.DoWork(); // Calls the new method.
```

```
BaseClass A = (BaseClass)B;
A.DoWork(); // Also calls the new method.
```

Virtual methods and properties allow you to plan ahead for future expansion. Because a virtual member is called regardless of which type the caller is using, it gives derived classes the option to completely change the apparent behavior of the base class.

Virtual members remain virtual indefinitely, no matter how many classes have been declared between the class that originally declared the virtual member. If class A declares a virtual member, and class B derives from A, and class C derives from B, class C inherits the virtual member, and has the option to override it, regardless of whether class B declared an override for that member. For example:

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

```
public class A
{
 public virtual void DoWork() { }
}
public class B : A
{
 public override void DoWork() { }
}

public class C : B
{
 public override void DoWork() { }
}
```

A derived class can stop virtual inheritance by declaring an override as sealed. This requires putting the sealed keyword before the **override** keyword in the class member declaration. For example:

```
public class C : B
{
 public sealed override void DoWork() { }
}
```

In the previous example, the method DoWork is no longer virtual to any class derived from C. It is still virtual for instances of C, even if they are cast to type B or type A. Sealed methods can be replaced by derived classes using the **new** keyword, as the following example shows:

```
public class D : C
{
 public new void DoWork() { }
}
```

In this case, if DoWork is called on D using a variable of type D, the new DoWork is called. If a variable of type C, B, or A is used to access an instance of D, a call to DoWork will follow the rules of virtual inheritance, routing those calls to the implementation of DoWork on class C.

A derived class that has replaced or overridden a method or property can still access the method or property on the base class using the base keyword. For example:



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

```
public class A
{
 public virtual void DoWork() { }
}
public class B : A
{
 public override void DoWork() { }
}

public class C : B
{
 public override void DoWork()
 {
 // Call DoWork on B to get B's behavior:
 base.DoWork();

 // DoWork behavior specific to C goes here:
 // ...
 }
}
}
```

### **Type conversion and casting**

When ever smaller datatype is assigned to larger datatypes then conversion is performed automatically. Known as **widening conversion**.

Eg:   int l;  
      byte b;  
      l = b //automatic conversion

When larger datatype is assigned to smaller type then casting should be performed known as **narrowing conversion**.

Eg:   int i;  
      byte b;  
      b = (byte)i; //casting

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT I**

**Subject Code : SCSX1008**

**Boxing & Unboxing**

**Boxing:**

Any type value a reference can be assigned to an object without an explicit conversion is known as Boxing.

Eg:    int i = 10;  
      object o = i;   //Boxing

**UnBoxing:**

Unboxing is the process of converting the object type back to value type.

Eg:    int i1 = (int)o;

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

*Assemblies*  
*Versioning*  
*Attributes*  
*Reflection*  
*Viewing metadata*  
*Type discovery – Reflecting on a type*  
*Marshaling*  
*Remoting – Understanding server object types – Specifying a server with an interface*  
*Building a server*  
*Building the client*  
*Exception handling*  
*Garbage collector.*

## **ASSEMBLIES**

Assemblies form the fundamental unit of deployment, version control, reuse, activation scoping, and security permissions for a .NET-based application. Assemblies take the form of an executable (.exe) file or dynamic link library (.dll) file, and are the building blocks of the .NET Framework. They provide the common language runtime with the information it needs to be aware of type implementations.

Assemblies can contain one or more modules. For example, larger projects may be planned in such a way that several individual developers work on separate modules, all coming together to create a single assembly.

Assemblies have the following properties:

- Assemblies are implemented as .exe or .dll files.
- Assembly can be shared between applications by putting it in the global assembly cache. Assemblies must be strong-named before they can be included in the global assembly cache.
- Assemblies are only loaded into memory if they are required. If they are not used, they are not loaded. This means that assemblies can be an efficient way to manage resources in larger projects.
- Obtain information about an assembly by using reflection. For more information, see Reflection.
- If you want to load an assembly only to inspect it, use a method such as ReflectionOnlyLoadFrom.

Two Types : **Private Assemblies**, used for single programs, and  
**Global Assemblies** shared among several applications.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

**Private Assemblies**

Intended use by single applications. Building modules to group common functionality.

- Location is specified at compile time
- PATH is not checked while looking up files, neither set by Control Panel 'System' configuration nor set in a Console Window.
- Identified by name and version if required. But only one version at a time.
- Digital signature possible to ensure that it can't be tampered.
- Get smaller EXE files.
- Dynamic linking, i.e. loading on demand.

**Global Assemblies**

Publicly sharing functionality among different application.

- Located in Global Assembly Cache (GAC).
- Identified by globally unique name and version.
- Digital signature to ensure that it can't be tampered.
- Get smaller EXE files.
- Dynamic linking, i.e. loading on demand.

**View Assemblies - The Intermediate Language Disassembler (ILDASM)**

Display metadata of one of your .NET programs or libraries by the use of the ILDASM tool:

```
C:\SS\> ildasm app1.exe
```

We can see the assembly's metadata with all the methods and types in a tree representation. If we click on 'M A N I F E S T' we get a window, which shows the manifest information which is shown in Fig 2.1 and 2.2.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

Subject Name : C# AND .NET

UNIT II

Subject Code : SCSX1008



Figure 2.1:Manifest Information

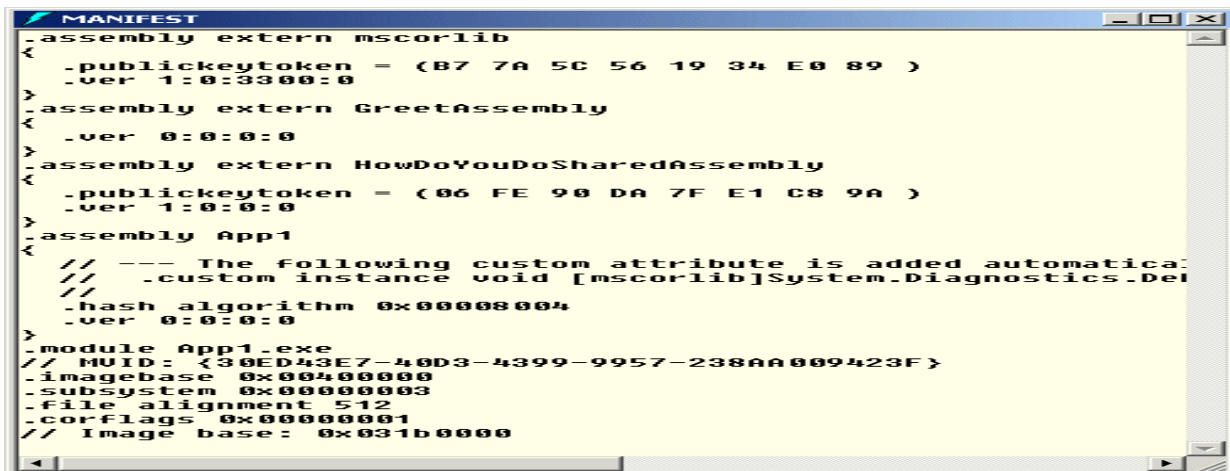


Figure 2.2:Manifest Information










Table 2.1 Graphic symbols used in Assembly

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

| Symbol                                                                              | Meaning       |
|-------------------------------------------------------------------------------------|---------------|
|    | Namespace     |
|    | Class         |
|    | Interface     |
|    | Enum          |
|    | Method        |
|    | Static method |
|    | Field         |
|    | Event         |
|  | Property      |

### **CREATE PRIVATE ASSEMBLY**

Consider :

#### **Hello.cs**

There is a simple class providing a method to print out a 'Hello':

```
namespace csharp.test.app.greet
{
 public class Hello {
 public void SayHello() {
 System.Console.WriteLine("Hello my friend, I am a DLL");
 }
 }
}
```

#### **GoodBye.cs**

Similar to Hello.cs but prints a 'Good bye':

```
namespace csharp.test.app.greet
{
 public class GoodBye {
 public void SayGoodBye() {
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

```
 System.Console.WriteLine("Good bye, I am a DLL too");
 }
}
```

Hello.cs and GoodBye.cs will be put into a single Private Assembly. They must be in the same namespace.

### **HowDoYouDo.cs**

We are going to implement this source file as a Global Assembly:

using System.Reflection;

```
[assembly:AssemblyKeyFile("app.snk")] //attributes
```

```
[assembly:AssemblyVersion("1.0.0.0")] //attributes
```

```
namespace csharp.test.app
```

```
{
```

```
 public class HowDoYouDo {
```

```
 public void SayHowDoYouDo() {
```

```
 System.Console.WriteLine("How do you do, I am a Global Assembly");
```

```
 }
```

```
 }
```

```
}
```

With the Attributes at the top we specify the key file used to generate a hash code and to declare the version.

### **Compile Classes to DLLs - The CSharp Compiler (CSC)**

To compile our source files we use the C# Compiler (csc):

```
DotNet> csc /debug /t:module /out:bin\Hello.dll Hello.cs
```

```
DotNet> csc /debug /t:module /out:bin\GoodBye.dll GoodBye.cs
```

```
DotNet> csc /debug /t:module /out:bin\HowDoYouDo.dll HowDoYouDo.cs
```

- the /debug includes debug information.
- the /t (target) switch lets us create a DLL.
- We are writing all our DLLs into a bin folder.





**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

The GAC can be found always in the 'assembly' sub folder inside %SystemRoot%, i.e. WINNT for Windows 2000 and Windows NT.

## **VERSIONING**

Version information for an assembly consists of the following four values:

- Major Version
- Minor Version
- Build Number
- Revision

We can specify all the values or can default the Build and Revision Numbers by using the '\*' as shown below:

[assembly: AssemblyVersion("1.0.\*")]

- The build number is the number of days since 01/01/2000
- The revision number is the number of 2 seconds periods since 00:00 of this day

This feature is great if we need some atomic versioning of a particular library for instance. Each program using this library has its own version and don't risk to break working feature in future release of this library. And if we want to force the use of a newer version of a library within a particular application, we can use some assembly redirection.

```
using System;
using System.Reflection;
[assembly:AssemblyVersion("1.1.0.0")]
class Example
{
 static void Main()
 {
 Console.WriteLine("The version of the currently executing assembly is: {0}",
 Assembly.GetExecutingAssembly().GetName().Version);

 Console.WriteLine("The version of mscorlib.dll is: {0}",
 typeof(String).Assembly.GetName().Version);
 }
}
```

### **Output :**

The version of the currently executing assembly is: 1.1.0.0  
The version of mscorlib.dll is: 2.0.0.0

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

## **ATTRIBUTES**

An attribute is a mechanism to add declarative information to code elements (types, members, assemblies or modules) beyond the usual predefined keywords. They are saved with the metadata of the object and can be used to describe the code at runtime or to affect application behaviour at run time through the use of reflection.

An *attribute* is an object that represents data you want to associate with an element in your program. The element to which you attach an attribute is referred to as the *target* of that attribute.

### **Intrinsic Attributes**

Attributes come in two flavors: *intrinsic* and *custom*. *Intrinsic* attributes are supplied as part of the Common Language Runtime (CLR), and they are integrated into .NET. *Custom* attributes are attributes you create for your own purposes.

Most programmers will use only intrinsic attributes, though custom attributes can be a powerful tool when combined with reflection.

### **Attribute Targets**

If you search through the CLR, you'll find a great many attributes. Some attributes are applied to an assembly, others to a class or interface, and some, such as [WebMethod], to class members. These are called the *attribute targets*. Possible attribute targets are given in Table 2.2.

| <b>Table 2.2 Possible attribute targets</b> |                                                                                                                                                                                |
|---------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Member Name</b>                          | <b>Usage</b>                                                                                                                                                                   |
| All                                         | Applied to any of the following elements: assembly, class, class member, delegate, enum, event, field, interface, method, module, parameter, property, return value, or struct |
| Assembly                                    | Applied to the assembly itself                                                                                                                                                 |
| Class                                       | Applied to instances of the class                                                                                                                                              |
| ClassMembers                                | Applied to classes, structs, enums, constructors, methods, properties, fields, events, delegates, and interfaces                                                               |
| Constructor                                 | Applied to a given constructor                                                                                                                                                 |
| Delegate                                    | Applied to the delegated method                                                                                                                                                |
| Enum                                        | Applied to an enumeration                                                                                                                                                      |
| Event                                       | Applied to an event                                                                                                                                                            |
| Field                                       | Applied to a field                                                                                                                                                             |

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

|             |                                                          |
|-------------|----------------------------------------------------------|
| Interface   | Applied to an interface                                  |
| Method      | Applied to a method                                      |
| Module      | Applied to a single module                               |
| Parameter   | Applied to a parameter of a method                       |
| Property    | Applied to a property (both get and set, if implemented) |
| ReturnValue | Applied to a return value                                |
| Struct      | Applied to a struct                                      |

### **Applying Attributes**

You apply attributes to their targets by placing them in square brackets immediately before the target item. You can combine attributes, either by stacking one on top of another:

```
[assembly: AssemblyDelaySign(false)]
[assembly: AssemblyKeyFile(".\\keyFile.snk")]
```

or by separating the attributes with commas:

```
[assembly: AssemblyDelaySign(false),
assembly: AssemblyKeyFile(".\\keyFile.snk")]
```

The key fact about intrinsic attributes is that you know when you need them; the task will dictate their use.

### **Custom Attributes**

You are free to create your own custom attributes and use them at runtime as you see fit. Suppose, for example, that your development organization wants to keep track of bug fixes. You already keep a database of all your bugs, but you'd like to tie your bug reports to specific fixes in the code.

You might add comments to your code along the lines of:

```
// Bug 323 fixed by Jesse Liberty 1/1/2005.
```

This would make it easy to see in your source code, but there is no enforced connection to Bug 323 in the database. A custom attribute might be just what you need. You would replace your comment with something like this:

```
[BugFixAttribute(323,"Jesse Liberty","1/1/2005")]
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

```
Comment="Off by one error"]
```

You could then write a program to read through the metadata to find these bug-fix notations and update the database. The attribute would serve the purposes of a comment, but would also allow you to retrieve the information programmatically through tools you'd create.

### **Declaring an Attribute**

Attributes, like most things in C#, are embodied in classes. To create a custom attribute, you derive your new custom attribute class from System.Attribute:

```
public class BugFixAttribute : System.Attribute
```

You need to tell the compiler with which kinds of elements this attribute can be used (the attribute target). You specify this with (what else?) an attribute:

```
[AttributeUsage(AttributeTargets.Class |
 AttributeTargets.Constructor |
 AttributeTargets.Field |
 AttributeTargets.Method |
 AttributeTargets.Property,
 AllowMultiple = true)]
```

AttributeUsage is an attribute applied to attributes: a meta-attribute. It provides, if you will, meta-metadata--that is, data about the metadata. For the AttributeUsage attribute constructor, you pass two arguments. The first argument is a set of flags that indicate the target--in this case, the class and its constructor, fields, methods, and properties. The second argument is a flag that indicates whether a given element might receive more than one such attribute. In this example, AllowMultiple is set to true, indicating that class members can have more than one BugFixAttribute assigned.

### **Naming an Attribute**

The new custom attribute in this example is named BugFixAttribute. The convention is to append the word Attribute to your attribute name. The compiler supports this by allowing you to call the attribute with the shorter version of the name. Thus, you can write:

```
[BugFix(123, "Jesse Liberty", "01/01/05", Comment="Off by one")]
```

The compiler will first look for an attribute named BugFix and, if it does not find that, will then look for BugFixAttribute.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

### **Constructing an Attribute**

Every attribute must have at least one constructor. Attributes take two types of parameters, *positional* and *named*. In the BugFix example, the programmer's name and the date are positional parameters, and comment is a named parameter. Positional parameters are passed in through the constructor and must be passed in the order declared in the constructor:

```
public BugFixAttribute(int bugID, string programmer,
string date)
{
 this.bugID = bugID;
 this.programmer = programmer;
 this.date = date;
}
```

**Named parameters are implemented as properties:**

```
public string Comment
{
 get
 {
 return comment;
 }
 set
 {
 comment = value;
 }
}
```

It is common to create read-only properties for the positional parameters :

```
public int BugID
{
 get
 {
 return bugID;
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

### **Using an Attribute**

Once you have defined an attribute, you can put it to work by placing it immediately before its target. To test the BugFixAttribute of the preceding example, the following program creates a simple class named MyMath and gives it two functions. You'll assign BugFixAttributes to the class to record its code-maintenance history:

```
[BugFixAttribute(121,"Jesse Liberty","01/03/05")]
[BugFixAttribute(107,"Jesse Liberty","01/04/05",
 Comment="Fixed off by one errors")]
public class MyMath
```

These attributes will be stored with the metadata. [Example 18-1](#) shows the complete program.

### **Example 18-1: Working with custom attributes**

```
namespace Programming_CSharp
{
 using System;
 using System.Reflection;

 // create custom attribute to be assigned to class members
 [AttributeUsage(AttributeTargets.Class |
 AttributeTargets.Constructor |
 AttributeTargets.Field |
 AttributeTargets.Method |
 AttributeTargets.Property,
 AllowMultiple = true)]
 public class BugFixAttribute : System.Attribute
 {
 // attribute constructor for
 // positional parameters
 public BugFixAttribute
 (int bugID,
 string programmer,
 string date)
 {
 this.bugID = bugID;
 this.programmer = programmer;
 this.date = date;
 }
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

```
// accessor
public int BugID
{
 get
 {
 return bugID;
 }
}

// property for named parameter
public string Comment
{
 get
 {
 return comment;
 }
 set
 {
 comment = value;
 }
}

// accessor
public string Date
{
 get
 {
 return date;
 }
}

// accessor
public string Programmer
{
 get
 {
 return programmer;
 }
}

// private member data
private int bugID;
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

```
private string comment;
private string date;
private string programmer;
}

// ***** assign the attributes to the class *****

[BugFixAttribute(121,"Jesse Liberty","01/03/05")]
[BugFixAttribute(107,"Jesse Liberty","01/04/05",
 Comment="Fixed off by one errors")]
public class MyMath
{

 public double DoFunc1(double param1)
 {
 return param1 + DoFunc2(param1);
 }

 public double DoFunc2(double param1)
 {
 return param1 / 3;
 }

}

public class Tester
{
 public static void Main()
 {
 MyMath mm = new MyMath();
 Console.WriteLine("Calling DoFunc(7). Result: {0}",
 mm.DoFunc1(7));
 }
}
}
```



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

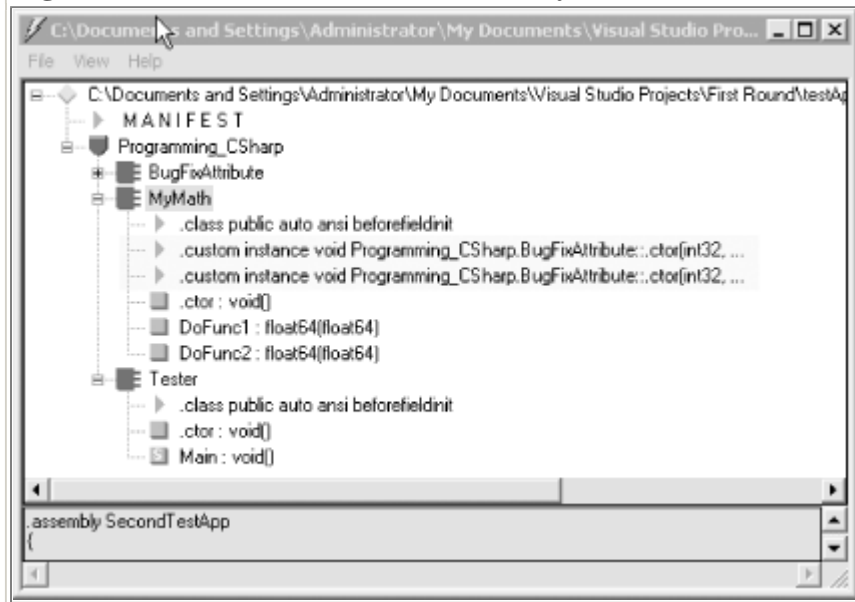
**Subject Code : SCSX1008**

**Output :**

Calling DoFunc(7). Result: 9.3333333333333339

As you can see, the attributes had absolutely no impact on the output. In fact, for the moment, you have only my word that the attributes exist at all. A quick look at the metadata using ILDasm does reveal that the attributes are in place, however, as shown in Figure 2.4.

**Figure 2.4. The metadata in the assembly**



**REFLECTION**

The classes in the Reflection namespace, along with the **System.Type** and **System.TypedReference** classes, provide support for examining and interacting with the metadata.

Reflection is generally used for any of **four tasks**:

**Viewing metadata :**

This might be used by tools and utilities that wish to display metadata.

**Performing type discovery :**

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

This allows you to examine the types in an assembly and interact with or instantiate those types. This can be useful in creating custom scripts. For example, you might want to allow your users to interact with your program using a script language, such as JavaScript, or a scripting language you create yourself.

**Late binding to methods and properties :**

This allows the programmer to invoke properties and methods on objects dynamically instantiated based on type discovery. This is also known as dynamic invocation.

**Creating types at runtime (Reflection Emit) :**

The ultimate use of reflection is to create new types at runtime and then to use those types to perform tasks. You might do this when a custom class, created at runtime, will run significantly faster than more generic code created at compile time.

**VIEWING METADATA :**

```
public static void Main()
{
 // get the member information and use it to
 // retrieve the custom attributes
 System.Reflection.MemberInfo inf = typeof(MyMath);
 object[] attributes;
 attributes =
 inf.GetCustomAttributes(
 typeof(BugFixAttribute), false);

 // iterate through the attributes, retrieving the
 // properties
 foreach(Object attribute in attributes)
 {
 BugFixAttribute bfa = (BugFixAttribute) attribute;
 Console.WriteLine("\nBugID: {0}", bfa.BugID);
 Console.WriteLine("Programmer: {0}", bfa.Programmer);
 Console.WriteLine("Date: {0}", bfa.Date);
 Console.WriteLine("Comment: {0}", bfa.Comment);
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

**Output:**

BugID: 121 Programmer: Jesse Liberty Date: 01/03/05 Comment:

BugID: 107 Programmer: Jesse Liberty Date: 01/04/05 Comment: Fixed off by one errors

**TYPE DISCOVERY REFLECTING**

**ON AN ASSEMBLY**

```
namespace Programming_CSharp
{
 using System;
 using System.Reflection;

 public class Tester
 {
 public static void Main()
 {
 // what is in the assembly
 Assembly a = Assembly.Load("mscorlib.dll");
 Type[] types = a.GetTypes();
 foreach(Type t in types)
 {
 Console.WriteLine("Type is {0}", t);
 }
 Console.WriteLine(
 "{0} types found", types.Length);
 }
 }
}
```

**Output :**

Type is System.TypeCode  
Type is System.Security.Util.StringExpressionSet  
Type is System.Runtime.InteropServices.COMException  
Type is System.Runtime.InteropServices.SEHException  
Type is System.Reflection.TargetParameterCountException  
Type is System.Text.UTF7Encoding  
Type is System.Text.UTF7Encoding+Decoder  
Type is System.Text.UTF7Encoding+Encoder  
Type is System.ArgIterator  
1426 types found .....

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

This example obtained an array filled with the types from the Core Library and printed them one by one. The array contained 1,426 entries on my machine.

**REFLECTING ON A TYPE :**

```
namespace Programming_CSharp
{
 using System;
 using System.Reflection;

 public class Tester
 {
 public static void Main()
 {
 // examine a single object
 Type theType =
 Type.GetType(
 "System.Reflection.Assembly");
 Console.WriteLine(
 "\nSingle Type is {0}\n", theType);
 }
 }
}
```

Output:

Single Type is System.Reflection.Assembly

**MARSHALLING**

Marshaling is the process of creating a bridge between managed code and unmanaged code; it is the homer that carries messages from the managed to the unmanaged environment and reverse. It is one of the core services offered by the CLR (Common Language Runtime.).NET code are called “managed” because it is controlled (managed) by the CLR. Other code that is not controlled by the CLR is called unmanaged.

**Marshalling Types During Platform Invoke (P/Invoke) on the Microsoft .NET Compact Framework.**

Data type representations in the Microsoft® .NET Compact Framework differ from those in unmanaged code. Converting between the managed and unmanaged representations is called marshaling and is automatic for most simple data types.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

Marshaling is the act of taking data from the environment you are in and exporting it to another environment. In the context of .NET, marshaling refers to moving data outside of the app-domain you are in, somewhere else.

**Marshalling Value and Reference Types :**

Value Types : STACK

Reference Types : HEAP

Value types are marshaled to unmanaged code on the stack. Reference types are passed by address. This means a pointer is passed on the stack, and the pointer contains the address of the marshaled data on the heap.

**BLITTABLE** : A type is considered blittable if it has a common representation in managed and unmanaged code memory.

**NON-BLITTABLE** : Non-blittable types require custom marshaling to convert between the unmanaged and managed representations.

**Value Types**

The data types outlined in the table below are automatically marshaled by value using P/Invoke. The table also shows the unmanaged equivalents in C/C++.

Common Value Types that are Automatically Marshaled

| <b>C#</b> | <b>Visual Basic .NET</b> | <b>Native C/C++</b> | <b>Size (bits)</b> |
|-----------|--------------------------|---------------------|--------------------|
| int       | Integer                  | int                 | 32                 |
| short     | Short                    | short               | 16                 |
| bool      | Boolean                  | BYTE                | 8                  |
| char      | Char                     | WCHAR               | 16                 |

Take the following C/C++ function as an example. The function accepts three integer parameters and calculates their arithmetic mean:

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

```
extern "C" _declspec(dllexport) int mean(int x, int y, int z)
{
 return (x + y + z) / 3;
}
```

This method can be declared and called in a Smart Device application. The following code calculates the mean of 1, 3 and 5 (that is, 3) and displays it in a Message Box:

```
using System.Runtime.InteropServices;
.
.
.
[DllImport("MarshalByValueDemo.dll")]
extern static int mean(int x, int y, int z);
.
.
.
int avg = mean(1, 3, 5);
MessageBox.Show(String.Format("The mean is {0}", avg));
```

It is worth noting; only value types of 32 bits or less can be marshaled automatically. Long types (64-bit integer) and floating-point types (float and double) cannot be marshaled by value into unmanaged code. You should pass these values by reference.

### **Reference Types**

In the .NET Compact Framework, reference types are, by default, passed by reference. When parameters are passed by reference, a pointer to the parameters on the managed heap is passed to the unmanaged code.

Since the unmanaged code receives a pointer, it is possible for the method to modify the data held on the managed heap. The .NET Compact Framework also supports passing value types by reference, using the **ref** keyword in **C#** and **ByRef** in **Visual Basic .NET**.

The following example takes three double parameters, passed by reference, and returns the arithmetic mean through a fourth parameter, also passed by reference.

```
extern "C" _declspec(dllexport) void mean(double* x, double* y, double* z,
double* mean)
{
 *mean = (*x + *y + *z) / 3.0;
}
```

To call this from managed code:

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

```
[DllImport("MarshalByRefDemo.dll")]
extern static void mean(ref double x, ref double y, ref double z, ref
 double mean);
.
.
.
double x = 1.0;
double y = 3.0;
double z = 5.0;
double avg = 0.0;
mean(ref x, ref y, ref z, ref avg);
MessageBox.Show(String.Format("The mean is {0}", avg));
```

## **MARSHALLING ARRAYS**

In C/C++, arrays are representations as pointers to a contiguous region of memory with the array elements addressed as offsets from this pointer, starting with zero.

The marshaler in the .NET Compact Framework Common Language Runtime (CLR) ensures that managed arrays adhere to this format when passed to unmanaged code.

In this C/C++ example, an array is passed into a function and is searched to locate the smallest value:

```
extern "C" _declspec(dllexport) int MinArray(int* pData, int length)
{
 // Initialise minData to the first element of the pData Array
 int minData = pData[0];
 int pos;

 // Loop through the array
 for(pos = 1; pos < length; pos++)
 {
 // If the current element is less than minData,
 // set minData to the value of current element
 if(pData[pos] < minData)
 minData = pData[pos];
 }

 return minData;
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

}

The following code declares the above method and calls it from C#:

```
[DllImport("MarshalArray.dll")]
extern static int MinArray(int[] pData, int length);
.
.
.
int[] sampleData = int[] {5, 1, 3 };
int result = MinArray(sampleData, sampleData.Length);
MessageBox.Show(String.Format("Smallest integer is {0}, result));
```

### **NET REMOTING OVERVIEW**

.NET Remoting is a Microsoft application programming interface (API) for interprocess communication

.NET remoting enables you to build widely distributed applications easily, whether application components are all on one computer or spread out across the entire world. You can build client applications that use objects in other processes on the same computer or on any other computer that is reachable over its network. You can also use .NET remoting to communicate with other application domains in the same process.

To use .NET remoting to build an application in which two components communicate directly across an application domain boundary, you need to build only the following:

- A remotable object.
- A host application domain to listen for requests for that object.
- A client application domain that makes requests for that object.

Remoting is a framework built into Common Language Runtime (CLR) in order to provide developers classes to build distributed applications and wide range of network services.

**Remoting provides various features** : Object Passing, Proxy Objects, Activation, Stateless and Stateful Object, Lease Based LifeTime and Hosting of Objects in IIS.

The namespaces that one typically uses in C# distributed object applications are the following:

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
```



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

When using one of the major IDE's (Visual Studio or C# Builder), it is important to add the reference system.remoting.dll to the build if it is not already present.

In C#, using distributed objects does not require stubs or interfaces as in Java. The CLR provides full support for remote object calls. Using distributed objects does not depend on the system registry for information about the remote classes. This information is encapsulated in a .DLL file that must be added as a reference when compiling the client code.

Classes derived from System.MarshalByRefObject cause the distributed object system to generate proxy objects on the client that encapsulate the low-level socket protocol. When the client sends a message to a remote object, it is the proxy that processes this message and sends serialized information across the network. The same works in reverse when proxy objects de-serialize information that is returned from the server.

Channel objects are the mechanism used to transfer messages between client and server. The .NET framework provides two bidirectional channels:

System.Runtime.Remoting.Channels.http.HttpChannel and  
System.Runtime.Remoting.Channels.Tcp.TcpChannel.

The http channel uses SOAP (Simple Object Access Protocol) and the tcp channel uses a binary stream. This latter method is more efficient because it avoids the need to encode and decode SOAP messages.

A channel must be registered before it can be used. The ChannelServices class is used to accomplish this as follows:

ChannelServices.RegisterChannel(someChannel);

The general steps involved in writing a distributed application are summarized below :

### **Writing the Server**

1. Construct the server class.
2. Select a method for hosting the server object(s) on the server. Typically a short application is created that launches the server and makes the server object available to the client(s).
3. The server object typically waits for one or more client objects to communicate with it.

### **Writing the Client**

1. Identify the remote server object to the client.
2. Connect the server to the client through a channel.
3. The client must activate the remote object and create a reference to it.
4. Communication to the remote object(s), once activated, is similar to sending messages to local objects.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

Let us consider a simple client server application.

### **Remoting Object**

This is the object to be remotely access by network applications. The object to be accessed remotely must be derived by MarshalByRefObject and all the objects passed by value must be serializable.

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

namespace RemotingSamples
{
 public class RemoteObject : MarshalByRefObject
 {

 ///constructor
 public RemoteObject()
 {
 Console.WriteLine("Remote object activated");
 }

 ///return message reply
 public String ReplyMessage(String msg)
 {
 Console.WriteLine("Client : "+msg); //print given message on console
 return "Server : Yeah! I'm here";
 }
 }
}
```

### **Server**

This is the server application used to register remote object to be accessed by client application. First, of all choose channel to use and register it, supported channels are HTTP, TCP and SMTP. I have used here TCP. Then register the remote object specifying its type.

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

```
namespace RemotingSamples
{
 public class Server
 {
 ///constructor
 public Server()
 {
 }
 ///main method
 public static int Main(string [] args)
 {
 //select channel to communicate
 TcpChannel chan = new TcpChannel(8085);
 //register channel
 ChannelServices.RegisterChannel(chan);
 //register remote object
 RemotingConfiguration.RegisterWellKnownServiceType(
 Type.GetType("RemotingSamples.RemoteObject,object"),
 "RemotingServer",
 WellKnownObjectMode.SingleCall);

 //inform console
 Console.WriteLine("Server Activated");
 return 0;
 }
 }
}
```

### **Client**

This is the client application and it will call remote object method. First, of all client must select the channel on which the remote object is available, activate the remote object and then call proxy's object method return by remote object activation.

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using RemotingSamples;
```

```
namespace RemotingSamples
{
 public class Client
 {

```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

```
 ///constructor
 public Client()
 {
 }
 ///main method
 public static int Main(string [] args)
 {
 //select channel to communicate with server
 TcpChannel chan = new TcpChannel();
 ChannelServices.RegisterChannel(chan);
 RemoteObject remObject = (RemoteObject)Activator.GetObject(
 typeof(RemotingSamples.RemoteObject),
 "tcp://localhost:8085/RemotingServer");

 if (remObject==null)
 Console.WriteLine("cannot locate server");
 else
 remObject.ReplyMessage("You there?");
 return 0;
 }
} }
```

**Deployment :**

To deploy this distributed application, the following sequence of steps must be followed:

- 1.The remote object must be compiled as follows to generate remote object.dll which is used to generate server and client executable.

```
csc /t:library /debug /r:System.Runtime.Remoting.dll remoteobject.cs
```

- 2.The server must be compiled as follows to produce server.exe.

```
csc /debug /r:remoteobject.dll /r:System.Runtime.Remoting.dll server.cs
```

3. The client must be compiled as follows in order to produce client.exe

```
csc /debug /r:remoteobject.dll /r:System.Runtime.Remoting.dll client.cs
```

**EXCEPTION HANDLING :**

An exception handling is an error that occurs at runtime. Using c# exception handling, you can handle runtime errors in a structured and controlled manner. Exception handling streamlines error-handling by allowing your program to define a block of code, called an exception handler, that is executed automatically when an error occurs. It is not necessary to check the success or

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

failure of each specific operation or method call manually. If an error occurs, it will be processed by the exception handler.

**The System.Exception Class:**

In c#, exception are represented by classes. All exception classes must be derived from the built-in class **Exception**, which is part of **System** namespace. Thus all exception are subclasses of **Exception**.

From **Exception** are derived **SystemException** and **ApplicationException**. These support the two general categories of exception:

- Those generated by the c# runtime system
- Those generated by application program

C# defines built-in exception that are derived from **SystemException**. For example, when division-by-zero is attempted, a **DivideByZero** Exception is generated.

**Exception Handling Fundamentals:**

C# exception handling is managed via four keywords: **try**, **catch**, **throw** and **finally**.

Program statements that you want to monitor for exceptions are contained within a **try** block. If an exception occurs within the **try** block, it is thrown. Your code can catch exception using **catch** and handle it in some rational manner. To manually throw an exception, use the keyword **throw**. Any code that absolutely must be executed upon exiting from a **try** block is put in a **finally** block.

**Using try and catch:**

At the core of exception handling are **try** and **catch**. These keywords work together. You can't have a **try** without a **catch**, or a **catch** without a **try**. The general form is:

```
try{
 //block of code to monitor for errors
}
catch(ExcepType1 exOb) {
 //handler for ExcepType1
}
catch(ExcepType2 exOb) {
 //handler for ExcepType2
}
```

Here when the exception is thrown, it is caught by its corresponding **catch** statement, which then processes the exception. As the general form shows, there can be more than one **catch** statement associated with a **try**. The type of exception determines which catch statement is executed. If no exception is thrown, then a try block ends normally and all the catch statements are bypassed. The catch statements are executed only if an exception is thrown.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

Exception Example :

Following is a simple example that illustrates how to watch for and catch an exception:

using System;

class Except

```
{
public static void Main()
{
int x=Int 32 Parse(Console.ReadLine());
int y=Int 32 Parse(Console.ReadLine());
int[] a ={ 10,5,3,4};
try
{
int z=x\y;
Console.WriteLine(z);
int b=a[3]+a[4];
Console.WriteLine(b);
}
}
catch(Exception e)
{
Console.WriteLine("error");
}
}
```

Output:

x=15

y=3

z=5

Using Multiple Catch Statements:

You can associate more than one **catch** statement with a **try**. However each **catch** statement must catch a different type of exception. The general form of **multiple catch** statement is:

```
catch(arithmetic Exception e1)
{
}
catch(ArrayIndexOutOfBoundsException e2)
{
}
catch(Exception e3)
{
}
```

Following is a simple example that illustrates how to use multiple catch statements:

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

```
using System;
class Except
{
public static void Main()
{
int x=Int 32 Parse(Console.ReadLine());
int y=Int 32 Parse(Console.ReadLine());
int[] a ={ 10,15,20,3};
try
{
int z=x\y;
Console.WriteLine(z);
int b=a[3]+a[4];
Console.WriteLine(b);
}
catch(arithmetic Exception e1)
{
Console.WriteLine("Arithmetic Exception);
}
catch(ArrayIndexOutOfBoundsException e2)
{
Console.WriteLine("ArrayOutOfBoundException");
}
}
}
```

**Output:**

x=10

y=2

z=5

ArrayOutOfBoundException

**Throwing An Exception:**

It is possible to manually throw an exception by using the throw statement.its general form is shown here:

Throw expectOb;

The expectOb must be an object of an exception class derived from **Exception**.

Following is a simple example that illustrates the **throw** statement by manually throwing a

**DivideByZeroException:**

```
using System;
class Throwdemo
{
public static void Main()
{
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

```
try
{
 Console.WriteLine("Before throw");
 throw new DivideByZeroException();
}
catch (DivideByZeroException)
{
 Console.WriteLine("Exception caught");
}
Console.WriteLine("After try/catch block");
}
}
}
```

**Output:**

Before throw

Exception caught

After try/catch block

**Rethrowing an Exception:**

An exception caught by one **catch** statement can be rethrown so that it can be caught by an outer **catch**. The most likely reason to rethrow an exception is to allow multiple handlers access to the exception. To rethrow an exception, you simply specify **throw**, without specifying an exception. That is, you use the form of **throw**:

**throw**;

If you rethrow an exception, it will not be recaptured by the same **catch** statement. It will propagate to the next **catch** statement.

Following is a simple example that illustrates the rethrowing of an exception:

using System;

class rethrow

```
{
 Int x=0,y=0;
 {
 public void div()
 {
 try
 {
 int z=x/y;
 throw new DivideByZeroException();
 }
 catch
 {
 throw; //rethrow the exception
 }
 }
 }
}
```



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

```
}
}
}
class excep
{
public static void Main()
{
 rethrow r=new rethrow();
 try
 {
 r.div();
 }
 catch(DivideByZeroException e)
 {
 Console.WriteLine("Exception received");
 }
}
}
```

Output:

Exception received

Using finally:

Sometimes you will want to define a block of code that will execute when a **try/catch** block is left. Such types of circumstances are common in programming, and c# provides a convenient way to handle them using **finally** keyword. To specify a block of code execute when a **try/catch** block is exited, include a **finally** block at the end of a **try/catch** sequence. The general form of a **try/catch** that includes **finally** is shown here:

```
try
{
 catch(ExcepType1 exOb)
 {
 }
 Catch(ExcepType2 exOb)
 {
 }
 .
 .
 .
 finally
 {
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

The **finally** block will be executed whenever execution leaves a **try/catch** block, no matter what conditions cause it. That is, whether the **try** block ends normally or because of an exception, the last code executed is that defined by **finally**. The finally block is also executed if any code within the **try** block or any of its **catch** statements returns from the method.

Following is a simple example that illustrates the use of finally keyword:

```
using System;
class finally
{
 public static void Main()
 {
 int x=10,y=0;
 int[]a={ 10,15};
 try
 {
 int z=x/y;
 int b=a[1]+a[2];
 }
 catch(Arithmetic Exception e1)
 {
 Console.WriteLine("Divide by Zero");
 }
 finally
 {
 Console.WriteLine("Error");
 }
 }
}
```

**Output :**      Divide by Zero Error

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

**Commonly Used Exception:**

| <b>Exception</b>           | <b>Meaning</b>                                                                                           |
|----------------------------|----------------------------------------------------------------------------------------------------------|
| ArrayTypeMismatchException | Type of value being stored is incompatible with the type of the array.                                   |
| DivideByZeroException      | Division by zero attempted                                                                               |
| IndexOutOfRangeException   | Array index is out of bounds                                                                             |
| InvalidCastException       | A runtime cast is invalid                                                                                |
| OutOfMemoryException       | A call to new fails because insufficient free memory exists                                              |
| OverflowException          | An arithmetic overflow occurred                                                                          |
| NullReferenceException     | An attempt was made to operate on a null reference that is, a reference that does not refer to an object |
| StackOverflowException     | The stack was overrun                                                                                    |

**Garbage Collection**

**Object lifetime in C#**

- ☐ Memory allocation for an object should be made using the “new” keyword
- ☐ Objects are allocated onto the managed heap, where they are automatically deallocated by the runtime at “some time in the future”
- ☐ Garbage collection is automated in C#

**Note :** Allocate an object onto the managed heap using the new keyword and forget about it

**Object creation**

- ☐ When a call to new is made, it creates a CIL “newobj” instruction to the code module

```
public static int Main (string[] args)
{
 Car c = new Car(“Viper”, 200, 100); }
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

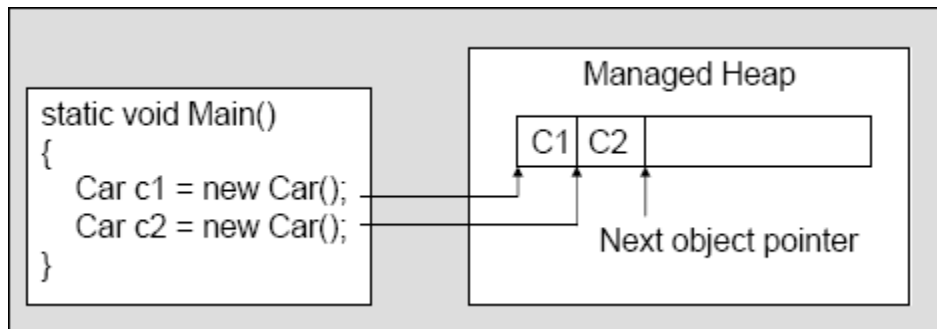
**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

**Tasks taken by CIL newobj instruction**

- ☐ Calculate the total amount of memory required for the object.
- ☐ Examine the managed heap to ensure enough room for the object.
- ☐ Return the reference to the caller, advance the next object pointer to point to the next available slot on the managed heap.



Rule: If the managed heap does not have sufficient memory to allocate a requested object, a garbage collection will occur.

**Garbage collection steps**

1. The garbage collector searches for managed objects that are referenced in managed Code - mark
2. The garbage collector attempts to finalize objects that are unreachable - Sweep
3. The garbage collector frees objects that are unmarked and reclaims their memory - Sweep

**Building finalizable objects**

```
//System.Object
public class Object
{
 ...
 protected virtual void Finalize() { }
}
```

- Override Finalize() to perform any necessary memory cleanup for your type
- A call to Finalize () occurs:
  - natural garbage collection
  - GC.Collect()
  - Application domain is unloaded from the memory

**The System.GC type**

- Provide a set of static method for interacting with garbage collection
- Use this type when you are creating types that make use of unmanaged resource.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Subject Name : C# AND .NET**

**UNIT II**

**Subject Code : SCSX1008**

**When to override System.Object.Finalize()**

The only reason to override Finalize() is if your C# class is making use of unmanaged resources via PInvoke or complex COM interoperability tasks (typically via the System.Runtime.InteropServices.Marshal type). It is illegal to override Finalize() on structure types.

**Building Disposable Objects**

- ✓ Another approach to handle an object's cleanup.
- ✓ Implement the IDisposable interface
- ✓ Object users should manually call Dispose() before allowing the object reference to drop out of scope
- ✓ Structures and classes can both support IDisposable (unlike overriding Finalize())

.

---

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

Interfaces

Collections

Enumerator

Cloneable objects

Comparable objects

Indexer

Delegates

Events

Multithreaded programming.

Programming with windows form controls – Windows form control Hierarchy

Adding controls – TextBox

CheckBoxes – RadioButtons – GroupBoxes

ListBoxes

3.13ComboBoxes

TrackBar

Calender

Spin Control

Panel

ToolTips

ErrorProvider

Dialog Boxes.

### **Interfaces**

Interfaces define properties, methods and events, which are the members of the interface. Interfaces contain only the declaration of the members. It is the responsibility of the deriving class to define the members. It often helps in providing a standard structure that the deriving classes would follow.

Abstract classes to some extent serve the same purpose, however, they are mostly used when only few methods are to be declared by the base class and the deriving class implements the functionalities.

- Interface support the concept of Multiple Inheritance.
- It is basically a kind of class with some differences
  1. All the members of an interface are implicitly public and abstract
  2. An interface cannot contain constant fields, constructors and destructors.
  3. Its members cannot be declared as static

### **Declaring Interfaces**

Interfaces are declared using the interface keyword. It is similar to class declaration. Interface statements are public by default. Following is an example of an interface declaration

Syntax:

```
interface interface_name
{
 member declarations;
}
ex:
interface Iname1
{
 void show();
}
```

### **Extending an interface**

Like classes, interface can also be extended. That is, an interface can be subinterfaced from other interfaces the new sub interface will inherit all the members of the superinterface.

Eg:

```
interface Iname2:iname1
{
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

**Implementing interfaces**

class may also inherit the interface known as Implementing interfaces.

Syntax:

```
class classname:interface name
{
 body of class
}
```

Now the class must implement all the methods of interface.

Suppose class want to inherit other classes and also interfaces means.

Syntax:

```
class classname:base class, interface1, interface2
{
}
```

name of each interface to be implemented must appear after the base class name.

**Program for declaring and implementing and interface**

```
using System;
interface IAdd
{
int add(); //implicitly this is public
}
interface IMul
{
int mul();
}
class A:IAdd,IMul
{
public int x,y;
public A(int x, int y)
{
this.x=x;
this.y=y;
}
public int add() // should be public
{
return(x+y);
}
public int mul()
{
return(x*y);
}
}
```



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

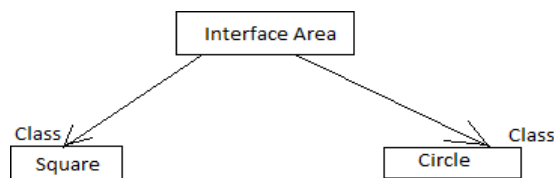
**Sub Code:SCSX1008**

```
class Ciface
{
public static void Main()
{
A a1 = new A(10,20);
Console.WriteLine(a1.add());
Console.WriteLine(a1.mul());
IAdd I1 = (IAdd) a1;
I1.add(); //call add method
Console.WriteLine(I1.add());
IMul I2 = (IMul) a1;
Console.WriteLine(I2.mul()); //call mul method
}
}
```

|                          |
|--------------------------|
| <b>Output:</b> 30 200 30 |
|--------------------------|

**Multiple implementation of an interface**

Same interface can be implemented by more than one classes.



**Program for multiple inheritance (implementation) of an interface**

```
using System;
interface IArea
{
double compute(double x);
}
class Square:IArea
{
public double compute(double x)
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
{
return(x*x);
}
}
class Circle:IArea
{
public double compute(double x)
{
return(3.14*x*x);
}
}
class Clifex
{
public static void Main()
{
IArea a1;
Square s1 = new Square();
Circle c1 = new Circle();
a1 = s1 as IArea; //casting as --> keyword
Console.WriteLine("Area of Square"+a1.compute(10));
a1 = c1 as IArea; //casting as --> keyword
Console.WriteLine("Area of circle"+a1.compute(10));
}
}
```

**Output**

Area of Square 100  
Area of Circle 314

**Explicit interface implementation**

Since Interface is user defined there might be a name collision , to avoid it we can explicitly implement an interface .

For eg. When a class implements two interfaces and both have same method signature (name,return type , no. of arguments) there will be a name collision so time we can use explicit interface implementation to avoid name collision.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

**Program for explicit interface implementation**

```
using System;
interface Iname1
{
 void display();
}
interface Iname2
{
 void display();
}
class C:Iname1,Iname2
{
 void Iname1.display() //no access modifier explicit interface and implementation.
 {
 Console.WriteLine("I1 display");
 }
 void Iname2.display() //no access modifier
 {
 Console.WriteLine("I2 display");
 }
}
class IFace
{
 public static void Main()
 {
 C c1 = new C();
 Iname1 I1 = (Iname1) c1;
 I1.display();
 Iname2 I2 = (Iname2) c1;
 I2.display();
 }
}
```

### Collections

In C#, a collection is a group of objects, the *System.Collections* namespace contains a large number of *interfaces and classes* that define and implement various types of collections.

#### Collection classes

Some of the general purpose collection classes are

1. ArrayList
2. Stack
3. Queue

#### Array List

The ArrayList class supports dynamic arrays, which can grow or shrink as needed. In C#, standard arrays are of a fixed length, which cannot be changed during program execution.

An ArrayList is a variable length array of object references that can dynamically increase or decrease in size.

An ArrayList is created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed the array can be shrunk.

#### **Program for Array List**

```
| using System;
 using System.Collections;
 class ArrayListDemo
 {
public static void Main()
{
 ArrayList a1 = new ArrayList();//ArrayList is created with the initial capacity 0
 Console.WriteLine("Initial number of element "+a1.Count);
 //count is the property used to count the number of elements in the ArrayList.
 a1.Add('A');
 a1.Add('B');
 a1.Add('C');
 a1.Add('D'); //Add elements to the ArrayList.
 Console.WriteLine("Current Content");
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
 for (int i=0; i<a1.Count;i++)
 Console.WriteLine(a1[i]); // print elements of ArrayList
 a1.Remove('B'); //Remove element from the ArrayList
 a1.Remove('C');
 Console.WriteLine("Number of elements "+a1.Count);
 for(int i=0; i<a1.Count;i++)
 Console.WriteLine(a1[i]);
 }
}
```

**Output :**

Initial number of elements 0

Current contents

A      B      C      D

Number of elements 2

A      C

**Queue**

Another familiar data structure is the queue, which is a first-in, first out list. (ie) first item put in a queue is the first item retrieved.

**Methods**

- public virtual void Enqueue(object v)
  - Adds v to the end of the queue
- public virtual object Dequeue()
  - **Returns the object at the front** of the invoking queue the **object is removed** in the process
- public virtual object Peek()
  - **Returns the object at the front** of the invoking queue, but **does not remove** it.
- Public virtual void Clear()
  - sets count to zero, which effectively clears the queue.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

**Example program for queue**

```
using System;
using System.Collections;
class QueueDemo
{
 public static void Main()
 {
 Queue q1 = new Queue();
 q1.Enqueue(55); //add object to the queue
 q1.Enqueue(65);
 q1.Enqueue(75);

 Console.WriteLine(q1.Dequeue());
 //returns the first element of the queue(55) and remove it
 Console.WriteLine(q1.Peek());
 //returns the first element of the queue(name it is 65) and
 //does not remove it
 Console.WriteLine(q1.Dequeue());
 }
}
```

**Output:**

```
55
65
65
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

**Stack**

- A stack is a first-in, last-out list

**Methods**

- public virtual void Push(object v)
  - pushes v into the stack
- public virtual object Pop()
  - Returns the element on the top of the stack, removing it in the process
- public virtual object Peek()
  - Returns the element on the top of the stack, but does not remove it
- public virtual void Clear()
  - used to clear the stack

**Program for stack**

```
using System;
using System.Collections;
class StackDemo
{
 public static void Main()
 {
 Stack s1 = new Stack();
 s1.Push(65); //push elements to the stack
 s1.Push(75);
 s1.Push(85);
 Console.WriteLine(s1.Pop());
 //now the last element in the top of the stack. Hence it retrieves the
 lastelement(85) and remove it
 Console.WriteLine(s1.Peek());
 // it retrieves the lastelement (now it is 75) and it does not remove it.
 Console.WriteLine(s1.Pop());
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
 }
}
```

**Output :**

85      75      75

### **Enumerator**

The Enumerators is an object that can return the elements of array, one by one in order, as they are requested. The enumerator “knows” the order of the items and keep track of where it is in the sequence. It then return the current item when it is requested.

#### **Example program for Enumerator**

**using System. ;**

**class Enum**

```
{
 public static void Main()
 {

 int[] Myarr={10,11,12,13};
 IEnumerator ie=Myarr.GetEnumerator();
 while(ie.MoveNext())
 {
 int i=(int) ie.Current;
 Console.WriteLine(i);
 }
 }
}
```



### **ICLONEABLE INTERFACE**

- The ICloneable interface contains one member, Clone, which is intended to support cloning beyond that supplied by MemberwiseClone.
- It is a procedure that can create a true, distinct copy of an object and all its dependent object, is to rely on the serialization features of the .NET framework.

There are two ways (types) to clone an instance:

- 1. Shallow copy** - may be linked to data shared by both the original and the copy
  - 2. Deep copy** - contains the complete encapsulated data of the original object
- The System.ICloneable interface defines a method of cloning—copying—to create a new instance of a class with the identical value as an existing instance.
  - A shallow copy is by far the easiest way to clone your class. This can be achieved with the MemberwiseClone method inherited by all classes from Object.

#### **Declaration syntax :**

```
public interface ICloneable
```

#### **Method syntax :**

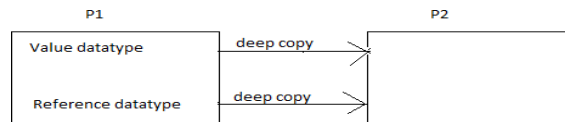
```
Public Object Clone ()
```

- The Clone method creates a new object—a copy of the current instance.
- Returns a new object that is a copy of the current instance.
- An Object of the same type as the current instance , containing copies of the non-static members of the current instance.
- The MemberwiseClone method creates a shallow copy by creating a new object, and then copying the nonstatic fields of the current object to the new object. If a field is a value type, a bit-by-bit copy of the field is performed. If a field is a reference type, the reference is copied but the referred object is not; therefore, the original object and its clone refer to the same object.

### **Deep Copy :**

Deep copy refers to a technique by which a copy of an object is created such that it contains copies of both instance members and the objects pointed to by reference members.

1. Deep copy is intended to copy all the elements of an object, which include directly referenced elements (of value type) and the indirectly referenced elements of a reference type that holds a reference (pointer) to a memory location that contains data rather than containing the data itself.
2. Deep copy is used in scenarios where a new copy (clone) is created without any reference to original data.



### **Program for Deep copy**

using System;

```
class Test
{
 public int a;
}
class Test1:ICloable
{
 int b;
 public Test o = new Test();
 public Test1(int x, int y)
 {
 o.a = x;
 b = y;
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
public void show()
{
 Console.WriteLine(o.a+"\t"+b);
}
public object Clone()//implementation of clone method for dupcopy
{
 Test1 temp = new Test1(o.a, b);
 return(temp);
}
}
class sample
{
 public static void Main()
 {
 Test e1 = new Test(10,20);
 Test e2 = e1.Clone();
 Console.WriteLine("Before changing");
 e1.show();
 e2.show();
 e1.o.a = 40;
 e1.b = 80;
 Console.WriteLine("After changing");
 e1.show();
 e2.show();
 }
}
```

**Output:**

*Before changing*

```
10 20 //for object e1
10 20 //for object e2
```

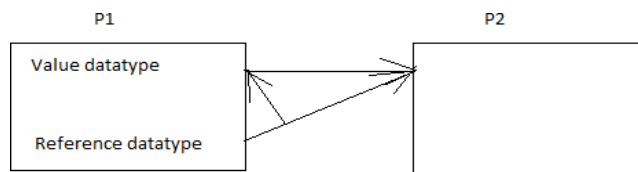
*After changing*

```
40 80 //for object e1
10 20 //for object e2
```

Since this is the deepcopy, changes in e1 object wont affect e2 object. Both can act as independently.

### Shallow Copy:

1. Shallow copy is the process of creating a clone of an object by instantiating a new instance of the same type as original object and copying the non-static members of the existing object to the clone.
2. The members of the value type are copied bit by bit while the members of the reference type are copied such that the referred object and its clone refer to the same object.
3. In general, shallow copy is used when performance is one of the requirements along with the condition that the object will not be mutated throughout the application. By passing the clone containing immutable data, the possibility of corruption by any code is eliminated.
4. Shallow copy is found to be efficient where object references allow objects to be passed around by memory address so that the entire object need not be copied.
5. Shallow copy is also known as memberwise copy.



### Program for shallow copy

```
using System;
class Test
{
 public int a;
}
class Test1: ICloneable
{
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
public Test O = new Test();
int b;
public Test1(int x, int y)
{
 o.a = x;
 b = y;
}
public void show()
{
 Console.WriteLine(o.a+"\t"+b);
}
public object Clone()// implementationof clone method for shallow copy
{
 return(this.MemberwiseClone());
}
}
class Example
{
 public static void Main()
 {
 Test1 e1 = new Test1(10,20);
 Test1 e2 = e1.Clone();//make e2 as the copy of e1
 Console.WriteLine("Before changing");
 e1.show();
 e2.show();
 e1.o.a = 40;
 e1.b = 80;
 Console.WriteLine("After Changing");
 e1.show();
 e2.show();
 }
}
```

```
 }
 }
O/P:
Before changing
10 20
10 20
After changing
40 80
40 20
```

### **3.3 ICOMPARABLE Interface**

The Icomparable interface specifies a behavior that allows an object to be sorted based on some internal key. It contains the method known as compareTo()

general form:-

```
int CompareTo(object o);
```

#### **CompareTo:**

Compares the current instance with another object of the same type and returns an integer that indicates whether the current instance precedes, follows, or occurs in the same position in the sort order as the other object.

#### **Program for Icomparable interface**

```
using System;
class Care:IComparable
{
 public int ID;
 public string name;
 public Car(int x, string name1)
 {
 name = name1;
 ID = x;
 }
 public int CompareTo(object o)
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
 {
 car temp = (car)o;
 if(this.ID > temp.ID)
 return(1);
 if(this.ID < temp.ID)
 return(-1);
 else
 return(0);
 }
 }
}
class car1
{
 public static void Main()
 {
 car[] c = new car[3];
 c[0] = new car("aaa",123);
 c[1] = new car("bbb", 12);
 c[2] = new car("ccc",1);
```

Array.sort(c); //sort an array of objects. Hence it will invoke built in sort function that  
will invoke compareTo method of class car for comparison

```
Console.WriteLine("After Sorting");
for each(car m in c)
 Console.WriteLine(m.name+"\t"+m.ID);
}
}
```

O/P:

```
ccc 1
bbb 12
aaa 123
```

### **Indexer**

Indexer is a pair of get and set accessors similar to those of properties.

It is possible to overload the [] operator for classes that you create, but you don't use in Operator method. Instead you create an indexer.

An indexer allows an object to be indexed like an array. Use is to support the creation of specialized arrays.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

Syntax for one dimensional indexer:

```
public return type this [int index]
{
 get
 {
 // return the value specified by the index
 }
 set
 {
 // set the value specified by the index
 }
}
```

Parameter index --> receives the index of the elements being accessed

get, set --> assessors. The assessors are called automatically when the indexer is used.

If the indexer is on the left side of an assignment statement, then set accessor is called and the value is set otherwise get accessor is called and the value is set otherwise get accessor is called and the value associated with the index must be returned.

[parameter index => receives the indexes of access element ]

Eg:

using System;

```
class IndexerEx
{
 // public int length;
 int[] a;
 public IndexerEx(int size) //constructor
 {
 a=new int[size];
 // length = size;
 }
}
```



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
 public int this[int id] // indexer
 {
 get
 {
 return(a[id]); //retrieve the value of an object array
 }
 set
 {
 a[id]=value;//set value for an object array
 }
 }
 }
}
class Exind
{
 public static void Main()
 {
 IndexerEx e1 = new IndexerEx(5);
 for(int i =0; i<5;i++)
 {
 e1[i] = i; // invokes set accessor of indexer
 Console.WriteLine(e1[i]); //invokes get accessor of indexer
 }
 }
}
```

**Output :**

Multidimensional indexer:-\

Syntax:

```
public RT this [int index1, int index2]
{
 get{ }
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
 set { }
 }
}
```

**Example:**

```
using System;
class TwoDInd
{
 public int r, c;
 int[,] a;
 public TwoDInd(int r1, int c1)
 {
 r = r1;
 c = c1;
 }
 public int this[int id1, int id2]
 {
 get
 {
 return(a[id1, id2]);
 }
 set
 {
 a[id1, id2] = value;
 }
 } //close indexer
} //close class
class Ex
{
 public static void Main()
 {
 twoDInd e1 = new TwoDInd(2,2);
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
 for(int j=0; j<2;j++)
 {
 e1[i,j] = i*j;//Invoke set block of an indexer
 Console.WriteLine(e1[i,j]);//invoke get block of an indexer
 }
 }
}
```

## **DELEGATES**

C# delegates are similar to pointers to functions, in C or C++. A **delegate** is a reference type variable that holds the reference to a method. The reference can be changed at runtime.

Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the **System.Delegate** class.

Syntax :

```
delegate <return type> <delegate-name> <parameter list>;
```

A delegate can refer to a method, which have the same signature as that of the delegate.

Steps:

- Create reference for the delegate
- create object for the delegate and the parameter is name of the desired methods
- Assign object to reference
- call the method through the reference

example:

```
// Create delegate
```

```
delegate void Del(String);
```

```
// Create a method for a delegate.
```

```
public static void DelegateMethod(string message)
```

```
{
```

```
 System.Console.WriteLine(message);
}
```

```
// Instantiate the delegate.
Del handler = DelegateMethod;
```

```
// Call the delegate.
handler("Hello World");
```

### Types of Delegates

- There are basically two types of **delegates**. **Single Cast** delegate and **Multi Cast** delegate.
- A **single cast** delegate can call only one function.
- A multi **cast** delegate is one that can be part of a linked list.
- The multi **cast** delegate points to the head of such a linked list. This means that when the multi **cast** delegate is invoked it can call all the functions that form a part of the linked list.
- To support a **single cast** delegate the base class library includes a special class type called `System.Delegate`. To support multi **cast delegates** the base class library includes a special class type called `System.MulticastDelegate`.

### Program for Single casting

```
using System;

namespace ConsoleApplication5
{
 class Program
 {
 public delegate void delmethod();

 public class P
 {
 public static void display()
 {
 Console.WriteLine("Hello!");
 }

 public static void show()
 {
 Console.WriteLine("Hi!");
 }
 }
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
public void print()
{
 Console.WriteLine("Print");
}

}

static void Main(string[] args)
{
 // here we have assigned static method show() of class P to delegate delmethod()
 delmethod del1 = P.show;

 // here we have assigned static method display() of class P to delegate delmethod() using
new operator
 // you can use both ways to assign the delegate
 delmethod del2 = new delmethod(P.display);

 P obj = new P();

 // here first we have create instance of class P and assigned the method print() to the
delegate i.e. delegate with class
 delmethod del3 = obj.print;

 del1();
 del2();
 del3();
}
}
}
```

**OUTPUT:**

Hi  
Hello  
Print

**Multicasting Delegates :**

This example demonstrates how to compose multicast delegates. A useful property of **delegate** objects is that they can be assigned to one delegate instance to be multicast using the **+**operator. A composed delegate calls the two delegates it was composed from. Only delegates of the same type can be composed.

The **-** operator can be used to remove a component delegate from a composed delegate.

### Example Program for multicasting delegates

```
delegate void Del(string s);

class TestClass
{
 static void Hello(string s)
 {
 System.Console.WriteLine(" Hello, {0}!", s);
 }

 static void Goodbye(string s)
 {
 System.Console.WriteLine(" Goodbye, {0}!", s);
 }

 static void Main()
 {
 Del a, b, c, d;

 // Create the delegate object a that references
 // the method Hello:
 a = Hello;

 // Create the delegate object b that references
 // the method Goodbye:
 b = Goodbye;

 // The two delegates, a and b, are composed to form c:
 c = a + b;

 // Remove a from the composed delegate, leaving d,
 // which calls only the method Goodbye:
 d = c - a;

 System.Console.WriteLine("Invoking delegate a:");
 a("A");
 System.Console.WriteLine("Invoking delegate b:");
 b("B");
 System.Console.WriteLine("Invoking delegate c:");
 c("C");
 System.Console.WriteLine("Invoking delegate d:");
 d("D");
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
}
}
```

Output

Invoking delegate a:

Hello, A!

Invoking delegate b:

Goodbye, B!

Invoking delegate c:

Hello, C!

Goodbye, C!

Invoking delegate d:

Goodbye, D!

### **Events**

Definition:

- An event is automatic notification when some action has occurred.
- An object that has an interest in an event, registers an event handler for that event.
- When the event occurs, all registered handlers are called.
- Event handlers are represented by delegates.
- Events are members of the class and are declared using the keyword 'event'.

General Form:

```
event event-delegate object;
 event-delegate-->Name of the delegate used to support the event
 object --> name of the specific event object.
```

Eg:

```
using System;
```

```
delegate void MyEventHandler();//Declare a delegate
```

```
//Declare an event class
```

```
class MyEvent
```

```
{
```

```
 public event MyEventHandler someEvent;
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
//It is called to fire the event
public void OnSomeEvent()
{
 if(SomeEvent != null)
 SomeEvent();
}
}
class EventDemo
{
 //An event handler
 static void handler()
 {
 Console.WriteLine("Event occurred");
 }
 public static void Main()
 {
 MyEvent e = new MyEvent();
 e.SomeEvent += new MyEventHandler(handler)
 e.OnSomeEvent();
 }
}
```

**Multicasting Events :**

More than object can receive event notification.

```
using System;
```

```
delegate void MyEventHandler();
```

```
class EC
```

```
{
```

```
 public event MyEventHandler SomeEvent;
```

```
 public void OnSomeEvent()
```

```
 {
```



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
 if(SomeEvent != null)
 SomeEnvet();
 }
}
class EC1
{
 public void Handler1()
 {
 Console.WriteLine("Event1");
 }
}
class EC2
{
 public void Handler2()
 {
 Console.WriteLine("Event2");
 }
}
class EC3
{
 public static void Main()
 {
 EC e1 = new EC();
 EC1 e2 = new EC1();
 EC2 e3 = new EC2();
 e1.SomeEvent += new MyEventHandler(e1.Handler1);
 e1.SomeEvent += new MyEventHandler(e2.Handler2);
 e1.OnSomeEvent();
 }
}
```

### **Multithreading**

A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution. Thus, multithreading is a specialized form of multitasking.

Multithreaded applications provide the illusion that numerous activities are happening at more or less the same time. But the reality is the CPU uses something known as “Time Slice” to switch between different threads.

The principal advantage of multithreading is that it enables you to write very efficient programs because it lets you utilize the idle time that is present in most programs. But using too many threads in our programs can actually “degrade” performance, as the CPU must switch between the active threads in the process which takes time. When a thread’s time slice is up, the existing thread is suspended to allow other thread to perform its business. For a thread to remember what was happening before it was kicked out of the way, it writes information to its local storage and it is also provided with a separate call stack, which again put extra load on the CPU.

All processes have at least one thread of execution, which is usually called the **main thread** because it is the one that is executed when your program begins. From the main thread you can create other threads.

The classes that support multithreaded programming are defined in the System.Threading namespace.

using System.Threading;

### **Creating a Thread**

To create a thread, you instantiate an object of type Thread. Thread defines the following constructor:

```
public Thread(ThreadStart entrypoint)
```

Here, entrypoint is the name of the method that will be called to begin execution of the thread. ThreadStart is a delegate defined by the .NET Framework as shown here:

```
public delegate void ThreadStart()
```

Thus, your entrypoint method must have a void return type and take no arguments.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

Once created, the new thread will not start running until you call its Start() method, which is defined by Thread. The Start() method is shown here:

```
public void Start()
```

Once started, the thread will run until the method specified by entryPoint returns. Thus, when entryPoint returns, the thread automatically stops. If you try to call Start() on a thread that has already been started, a ThreadStateException will be thrown.

**Example :**

```
using System;
using System.Threading;
namespace CSharpThreadExample
{
 class Program
 {
 public static void run()
 {
 for (int i = 0; i < 5; i++)
 {
 Console.WriteLine("In thread " + Thread.CurrentThread.Name + i);
 Thread.Sleep(1000);
 }
 }
 static void Main(string[] args)
 {
 Console.WriteLine("Main Thread Starting");
 Thread.CurrentThread.Name = "Main ";

 Thread t1 = new Thread(new ThreadStart(run));
 t1.Name = "Child";
 t1.Start();

 for (int i = 0; i < 5; i++)
 {
 Console.WriteLine("In thread " + Thread.CurrentThread.Name + i);
 Thread.Sleep(1000);
 }
 Console.WriteLine("Main Thread Terminates");
 Console.Read();
 }
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

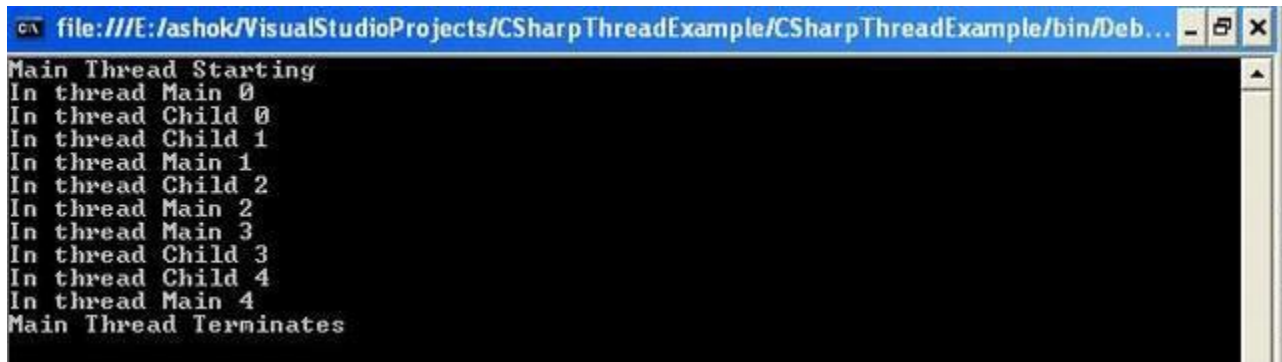
```
}
```

Notice the call to `Sleep()`, which is a static method defined by `Thread`. The `Sleep()` method causes the thread from which it is called to suspend execution for the specified period of milliseconds. The form used by the program is shown here:

```
public static void Sleep(int milliseconds)
```

The number of milliseconds to suspend is specified in milliseconds. If milliseconds is zero, the calling thread is suspended only to allow a waiting thread to execute.

Here's the output:



```
file:///E:/ashok/VisualStudioProjects/CSharpThreadExample/CSharpThreadExample/bin/Deb...
Main Thread Starting
In thread Main 0
In thread Child 0
In thread Child 1
In thread Main 1
In thread Child 2
In thread Main 2
In thread Main 3
In thread Child 3
In thread Child 4
In thread Main 4
Main Thread Terminates
```

Each thread maintains a private set of structures that the O/S uses to save information(the thread context) when the thread is not running including the value of CPU registers. It also maintains the priority levels. We can assign higher priority to a thread of more important task than to a thread which works in background. In all cases time slice allocated to each thread is relatively short, so that the end user has the perception that all the threads (and all the applications) are running concurrently.

O/S has thread scheduler which schedules existing threads and preempts the running thread when its time slice expires. We make the most use of multithreading when we allocate distinct threads to tasks that have different priorities or that take a lot of time to complete.

The main problem with threads is that they compete for shared resource, a resource can be a variable, a database connection, a H/W device. We must synchronize the access to such resources — otherwise we will result in deadlock situations. A thread terminates when the task provided to it terminates or when the thread is programmatically killed by calling `Abort()`. An application as a whole terminates only when all its threads terminate.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

### **Methods**

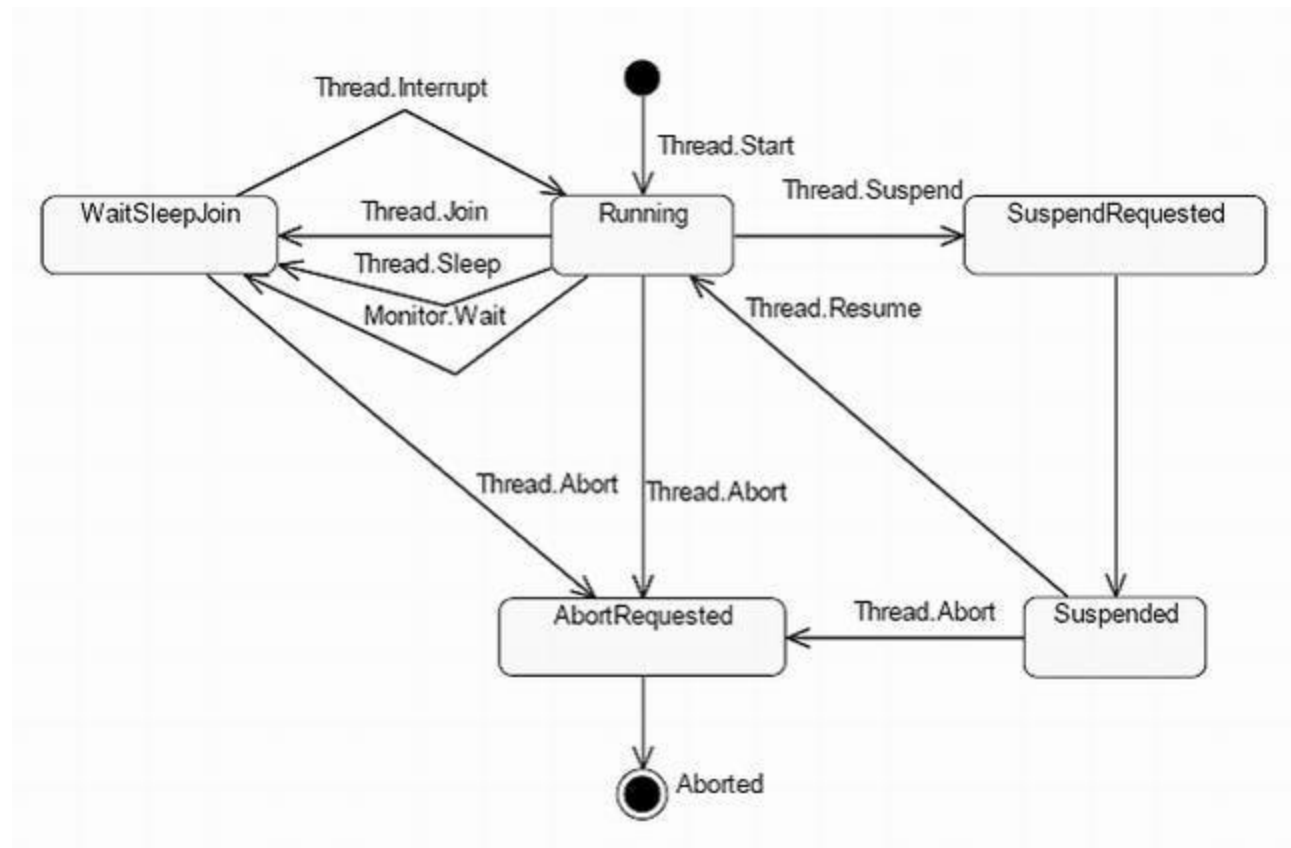
- Suspend() -> Suspends the execution of a thread till Resume() is called on that.
- Resume() -> Resumes a suspended thread. Can throw exceptions for bad state of the thread.
- Sleep() -> A thread can suspend itself by calling Sleep(). Takes parameter in form of milliseconds. We can use a special timeout 0 to terminate the current time slice and give other thread a chance to use CPU time
- Join()-> Called on a thread makes other threads wait for it till it finishes its task.

### **States of a Thread**

States of a thread can be checked using ThreadState enumerated property of the Thread object which contains a different value for different states.

- Aborted -> Aborted already.
- AbortRequested -> Responding to an Abort() request.
- Background -> Running in background. Same as IsBackground property.
- Running -> Running after another thread has called the start()
- Stopped -> After finishing run() or Abort() stopped it.
- Suspended -> Suspended after Suspend() is called.
- Unstarted -> Created but start() has not been called.
- WaitSleepJoin -> Sleep()/Wait() on itself and join() on another thread. If a thread Thread1 calls sleep() on itself and calls join() on the thread Thread2 then it enters WaitSleepJoin state. The thread exists in this state till the timeout expires or another thread invokes Interrupt() on it.

It is wise to check the state of a thread before calling methods on it to avoid ThreadStateException.



This picture describes in detail about the states of the thread [ Collected from “**Thinking in C#**” by **Bruce Eckel** ]

### Properties of a Thread

- **Thread.CurrentThread** -> Static method gives the reference of the thread object which is executing the current code.
- **Name** -> Read/Write Property used to get and set the name of a thread
- **ThreadState** -> Property used to check the state of a thread.
- **Priority** -> Property used to check for the priority level of a thread.
- **IsAlive** -> Returns a Boolean value stating whether the thread is alive or not.
- **IsBackground** -> Returns a Boolean value stating the running in background or foreground.

### **PriorityLevels of Thread**

Priority levels of thread is set or checked by using an enumeration i.e. ThreadPriority. The valid values are for this enumeration are;

- **Highest**
- **AboveNormal**
- **Normal**
- **BelowNormal**
- **Lowest**

### **Synchronization in Threads**

When we have multiple threads that share data, we need to provide synchronized access to the data. We have to deal with synchronization issues related to concurrent access to variables and objects accessible by multiple threads at the same time. This is controlled by giving one thread a chance to acquire a lock on the shared resource at a time. We can think it like a box where the object is available and only one thread can enter into and the other thread is waiting outside the box until the previous one comes out.

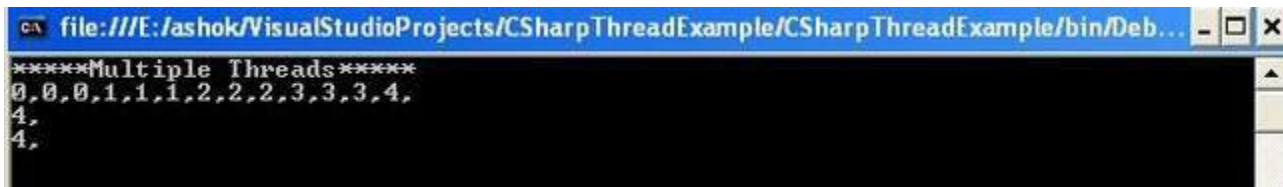
```
using System;
using System.Threading;
namespace CSharpThreadExample
```

```
{
 class Program
 {
 static void Main(string[] arg)
 {
 Console.WriteLine("*****Multiple Threads*****");
 Printer p=new Printer();
 Thread[] Threads=new Thread[3];
 for(int i=0;i<3;i++)
 {
 Threads[i]=new Thread(new ThreadStart(p.PrintNumbers));
 Threads[i].Name="Child "+i;
 }
 foreach(Thread t in Threads)
 t.Start();

 Console.ReadLine();
 }
 }
}
```

```
class Printer
{
 public void PrintNumbers()
 {
 for (int i = 0; i < 5; i++)
 {
 Thread.Sleep(100);
 Console.Write(i + ",");
 }
 Console.WriteLine();
 }
}
```

In the above example, we have created three threads in the main method and all the threads are trying to use the PrintNumbers() method of the same Printer object to print to the console. Here we get this type of output:



Now we can see, as the thread scheduler is swapping threads in the background each thread is telling the **Printer** to print the numerical data. We are getting inconsistent output as the access of these threads to the **Printer** object is synchronized. There are various synchronization options which we can use in our programs to enable synchronization of the shared resource among multiple threads.

### Using the Lock Keyword

In C# we use lock(object) to synchronize the shared object.

Syntax:

```
lock (objecttobelocked) {
 objecttobelocked.somemethod();
}
```

Here objecttobelocked is the object reference which is used by more than one thread to call the method on that object. The lock keyword requires us to specify a token (an object reference) that



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

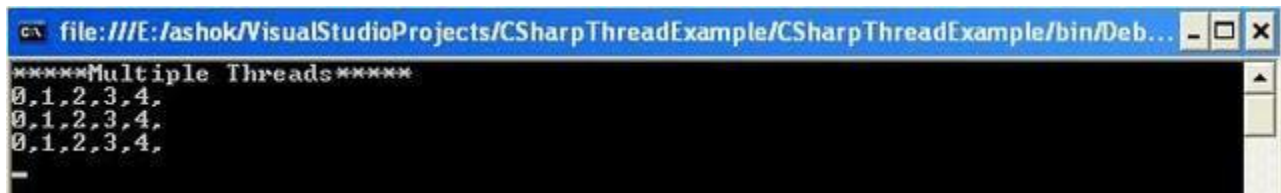
must be acquired by a thread to enter within the lock scope. When we are attempting to lock down an instance level method, we can simply pass the reference to that instance. (We can use this keyword to lock the current object) Once the thread enters into a lock scope, the lock token (object reference) is inaccessible by other threads until the lock is released or the lock scope has exited.

If we want to lock down the code in a static method, we need to provide the System.Type of the respective class.

### **Converting the Code to Enable Synchronization using the Lock Keyword**

```
public void PrintNumbers()
{
 lock (this)
 {
 for (int i = 0; i < 5; i++)
 {
 Thread.Sleep(100);
 Console.Write(i + ",");
 }
 Console.WriteLine();
 }
}
```

### **OUTPUT**



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

**Programming with Windows Form Controls**

The .NET platform provides three GUI tool kits:

1. Windows Forms
2. Web Forms
3. Mobile forms

System.Windows.Forms namespaces contains a number of types that allow you to build traditional desktop application.

System.Web.UI.WebControls namespace contain a number of such types that allow you to build the browser independent front ends.

System.Web.UI.MobileControls used to build UIs that target the mobile devices.

**System.Windows.Forms Namespace :**

| Windows Form Class                                        | Meaning                                                                            |
|-----------------------------------------------------------|------------------------------------------------------------------------------------|
| Application                                               | Using the members of Application, start and terminate a Windows Forms Application. |
| ButtonBase, Button, Checkbox, ComboBox, GroupBox, ListBox | These classes represent widgets used to build various GUI widgets.                 |
| Form                                                      | Represents main window, dialog box or MDI child window.                            |
| ColorDialog, OpenFileDialog, SaveFileDialog, FontDialog   | Used to create built-in dialog boxes.                                              |
| Menu, MainMenu, MenuItem, ContextMenu                     | Used to create Menu Systems.                                                       |
| Statubar, ToolBar, ScrollBar                              | Used to create Child Controls.                                                     |

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name: C# and .NET**

**UNIT III**

**Sub Code: SCSX1008**

**APPLICATION CLASS**

**Building a Window**

Steps for creating MainWindow:-

1. Derive the new custom class from System.Windows.Forms.Form
2. Configure the Application Main() method to call Application.Run() passing an instance of your new Form-derived type as an argument.

Program:

using System;

using System.Windows.Forms;

class WinForm:Form

{

    public static void Main()

    {

        Application.Run(new WinForm());

    }

}

Application class defines members that allow you to control various low-level behaviours of a Windows Forms application.

**Methods:**

|              |                                                                                             |
|--------------|---------------------------------------------------------------------------------------------|
| DoEvents()   | Provides the ability for an application to process messages currently in the message queue. |
| Exit()       | Terminates the Application                                                                  |
| Run()        | Begins running a standard application message loop on the current thread.                   |
| ExitThread() | Exits the message loop on the current thread                                                |

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

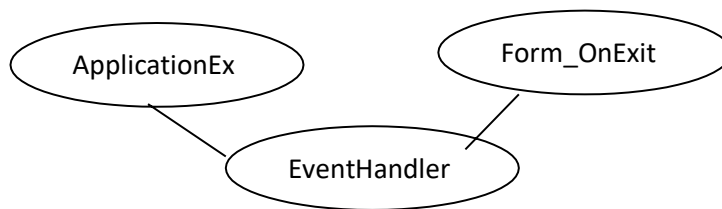
**Properties:**

|                      |                                                                          |
|----------------------|--------------------------------------------------------------------------|
| CompanyName          | Retrieves the company name associated with the current application.      |
| CurrentInputLanguage | Gets or sets the current input language for the current thread           |
| ProductName          | Retrieves the product name associated with the current application.      |
| ProductVersion       | Retrieves the product version associated with the current application.   |
| StartupPath          | Retrieves the path for the executable file that started the application. |

**Events:**

|                 |                                                                |
|-----------------|----------------------------------------------------------------|
| ApplicationExit | Occurs when the application is just about to shut down.        |
| ThreadExit      | Occurs when a thread in the application is about to terminate. |

If you want your form to respond to the ApplicationExit event, register an eventhandler using += operator.



***Syntax for Delegate :***

```
public void EventHandler(Object o,EventArgs e)
```

```
Application Exit -> Event
```

```
Form_OnExit -> EventHandler
```

```
EventHandler -> Delegate
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

*Points to Remember:*

- Register event handler to event using the delegate
- Can't change event & event delegate name.
- Implement the event handler, signature should be same as delegate signature.

**Example program for Application Properties & Application Exit**

```
using System.Windows.Forms;

using System;

class WinForm:Form

{

String str=" ";

public WinForm()

{

Application.ApplicationExit+=new EventHandler(Form_OnExit);

}

public void Form_OnExit(object o,EventArgs e)

{

str="Company Name :"+Application.CompanyName+"\n";

str+="Input Language :"+Application.CurrentInputLanguage+"\n";

str+="Name :"+Application.ProductName+"\n";

str+="Version :"+Application.ProductVersion+"\n";

str+="Path :"+Application.StartupPath;

MessageBox.Show(str);

}

public static void Main()

{

Application.Run(new WinForm());

}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

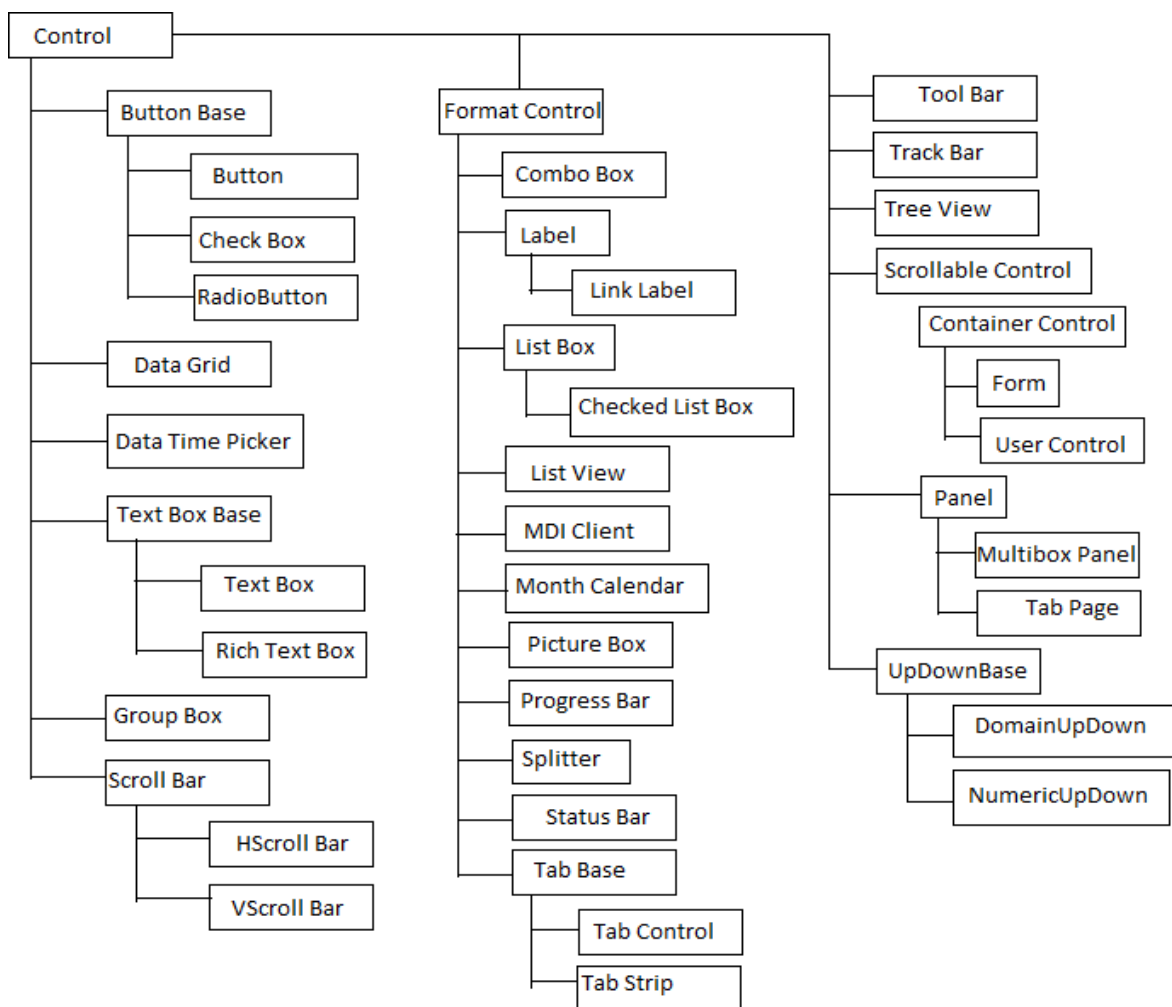
**UNIT III**

**Sub Code:SCSX1008**

}

}

**Windows Form Control Hierarchy**



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

**Adding Controls to Forms:**

3 steps:

1. Define member variables that represent the GUI widgets.
2. Assign properties for each GUI widget.
3. Add the GUI widget to the form.

Eg:

using System.Windows.Forms;

class WinForm:Form

```
{
 //Define member variables
 TextBox t1=new TextBox();
 public Winform()
 {
 //Assign properties
 t1.text="Save"
 //Add cntrols
 this.Controls.Add(t1);
 }
}
```

**Text Box**

It contains single or multiple lines of text base class of text box is TextBoxBase.

**Properties of TextBoxBase:**

- |                   |    |                                                                                                                                                                     |
|-------------------|----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. AcceptsTab     | => | Indicates if pressing the Tab key in a multiline TextBox control tabs within the control itself, rather than moving the focus to the next control in the tab order. |
| 2. BackColor      | => | Get or set the background color of the control                                                                                                                      |
| 3. ForeColor      | => | Get or set the foreground color of the control                                                                                                                      |
| 4. MaxLength      | => | Maximum number of characters that can be entered into the TextBox control                                                                                           |
| 5. Multiline      | => | Specifies if this TextBox can contain multiple lines of text                                                                                                        |
| 6. ReadOnly       | => | Marks this TextBox as read only                                                                                                                                     |
| 7. SelectedText   | => | Returns the selectedText                                                                                                                                            |
| 8. SelectedLength | => | Returns the length of the selected text                                                                                                                             |
| 9. SelectionStart | => | Gets or Sets the starting point of the selected text                                                                                                                |

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

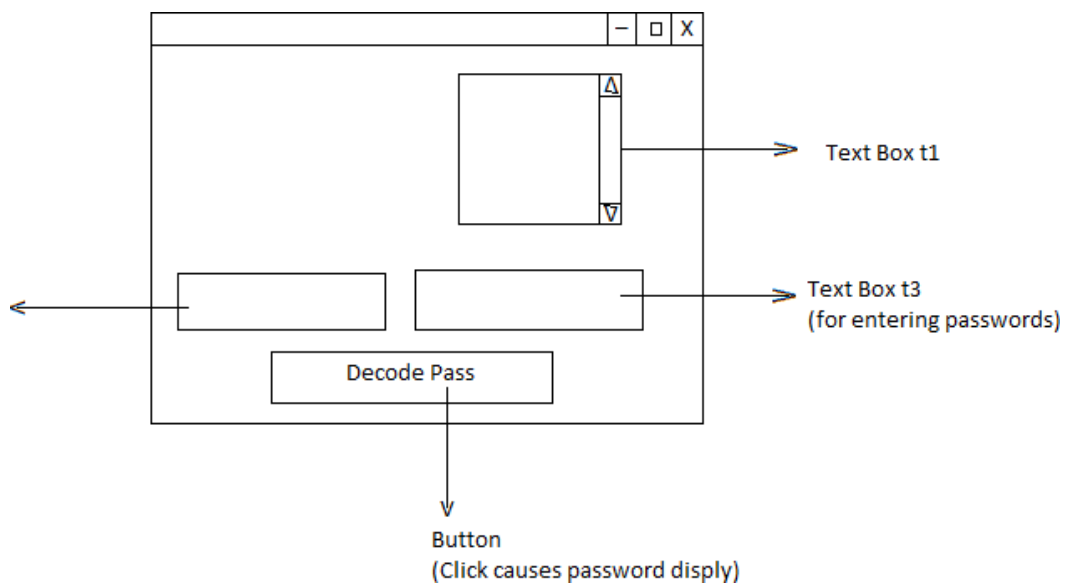
**UNIT III**

**Sub Code:SCSX1008**

TextBox Properties:

- |                    |    |                                                                                                               |
|--------------------|----|---------------------------------------------------------------------------------------------------------------|
| 1. CharacterCasing | => | Gets or Sets whether the TextBox control modifies the case of characters as they are typed.                   |
| 2. PasswordChar    | => | Gets or Sets the character used to mask characters in a single line Textbox. Control used to enter passwords. |
| 3. ScrollBars      | => | Gets or Sets which scroll bars should appear in a multiline TextBox control                                   |
| 4. TextAlign       | => | Gets or Sets how text is aligned in the text box control                                                      |

Eg: Design the form in the following manner



```
using System;
using System.Windows.Forms;
// using System.Windows.Forms.Controls;
class WinText:Form
{
 //Create reference for the required controls
 TextBox t1=new TextBox();
 TextBox t2=new TextBox();
```



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
TextBox t3=new TextBox();
Button b1=new Button();
public WinText()
{
 t1.Location=new System.Drawing.Point(70,10);
 t1.Size=new System.Drawing.Size(50,50);
 t1.AcceptsTab=true;
 t1.Multiline=true;
 t1.ScrollBars=ScrollBars.Vertical;
 t2.Location=new System.Drawing.Point(10,70);
 t2.Size=new System.Drawing.Size(50,20);
 // t2.CharacterCasing=CharacterCasing.UpperCase;
 t3.Location=new System.Drawing.Point(70,70);
 t3.Size=new System.Drawing.Size(50,20);
 // t3.PasswordChar='$';
 b1.Location=new System.Drawing.Point(110,90);
 b1.Size=new System.Drawing.Size(50,20);
 b1.Text= "Decode Password";
 this.Controls.Add(t1);
 this.Controls.Add(t2);
 // this.Controls.Add(t3);
 this.Controls.Add(b1);
 b1.Click+=new EventHandler(b1_Click);
}
// Implement the event handler OnClick

public void b1_Click(object o,EventArgs e)
{
 MessageBox.Show(t2.Text);
}
// Implement the main() method
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name: C# and .NET**

**UNIT III**

**Sub Code: SCSX1008**

```
public static void Main()
{
 Application.Run(new WinText());
}
}
```

**Check Box**

Represents set of possible items from which you can select more than one item.

**Properties:**

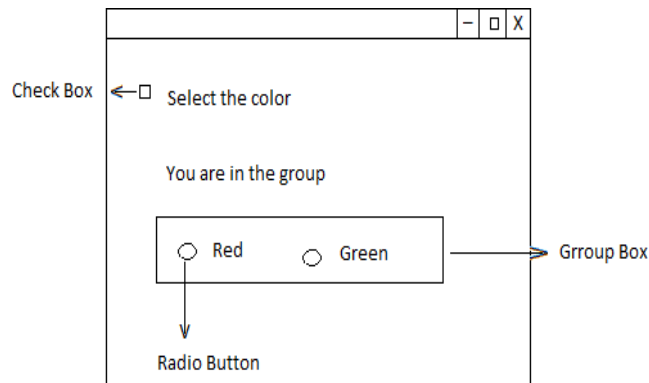
|            |   |                                                                                           |
|------------|---|-------------------------------------------------------------------------------------------|
| CheckAlign | : | Gets or sets the horizontal and vertical alignment of a check box on a check box control. |
| Checked    | : | Returns the boolean value representing the state of the checkbox.                         |
| CheckState | : | Gets or Sets the value indicating whether the checkbox is checked.                        |

### **RadioButtons & GroupBox**

More than one radio buttons can be grouped using the GroupBox control. From the Group, you can select only one radio button.

GroupBox Events: Enter, Leave

Eg: Design the form



```
using System.Windows.Forms;
using System.Windows.Forms.Control;
class WinForm:Form
{
 CheckBox c1=new CheckBox();
 RadioButton r1=new RadioButton();
 RadioButton r2=new RadioButton();
 GroupBox g1=new GroupBox();
 public WinForm()
 {
 c1.Location=new System.Drawing.Point(5,5);
 c1.Size=new System.Drawing.Size(70,20);
 c1.Text= "Select the colour";
 r1.Location=new System.Drawing.Point(30,30);
 r1.Size=new System.Drawing.size(30,20);
 r2.Location=new System.Drawing.Point(60,30);
 r2.Size=new System.Drawing.size(50,20);
 g1.Location=new System.Drawing.Point(25,30);
 g1.Controls.Add(r1);
 g1.Controls.Add(r2);
 this.Controls.Add(g1);
 this.Controls.Add(c1);
 g1.Enter+=new EventHandler(OnEnter);
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

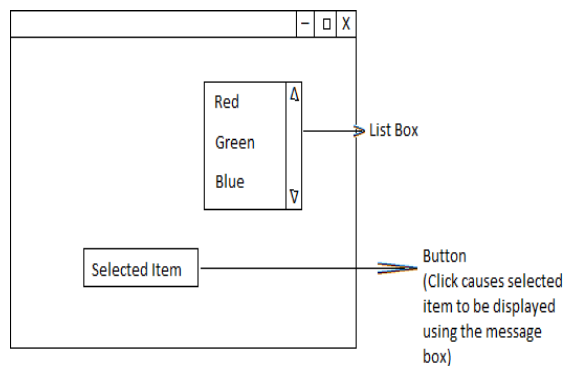
**Sub Code:SCSX1008**

```
public void OnEnter(object o, EventArgs e)
{
 g1.Text= "You are in the group";
}
public static void Main()
{
 Application.Run(new WinForm());
}
}
```

**List Box**

List box contains list of items from which you can select more than one item.

Design the form:



```
using System.Windows.Forms;
using System.Windows.Forms.Control;
class WinForm:Form
{
 ListBox l1=new ListBox();
 Button b1=new Button();
 public WinForm()
 {
 l1.Location=new System.Drawing.Point(60,20);
 l1.Size=new System.Drawing.Size(50,50);
 l1.Items.AddRange(new object[3]{ "Red", "Green", "Blue"});
 b1.Location=new System.Drawing.Point(20,80);
 b1.Size=new System.Drawing.Size(20,20);
 this.Controls.Add(l1);
 this.Controls.Add(b1);
 b1.Click+=new EventHandler(OnClick);
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

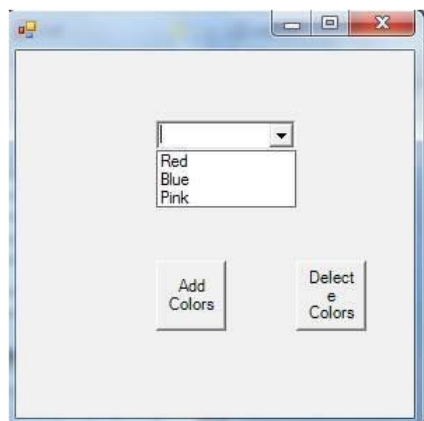
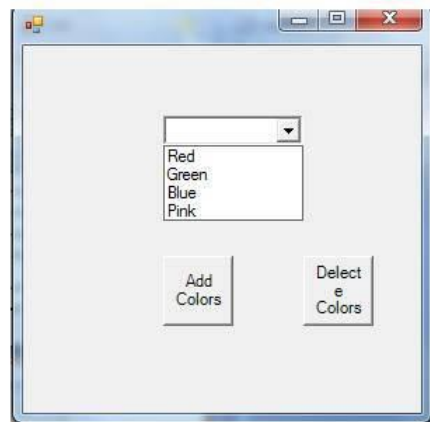
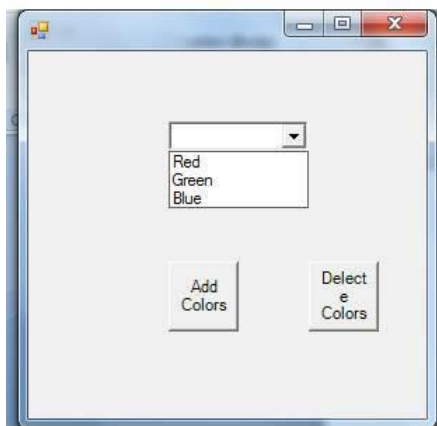
**Sub Code:SCSX1008**

```
 if(l1.SelectedItem!=null)
 MessageBox.Show(l1.SelectedItem);
 }
 public static void Main()
 {
 Application.Run(new WinForm());
 }
}
```

**Combo Box**

ComboBox type is unique in that the user can also insert additional items.

Ex: Design The following form



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
using System;
using System.Windows.Forms;
class com:Form
{
 ComboBox c = new ComboBox();
 Button b1 = new Button();
 Button b2 = new Button();
 public com()
 {
 c.Location= new System.Drawing.Point(100,50);
 c.Size= new System.Drawing.Size(100,50);
 c.Items.AddRange(new Object[] { "Red","Green","Blue"});
 b1.Location= new System.Drawing.Point(100,250);
 b1.Size= new System.Drawing.Size(50,50);
 b1.Text="Add Colors";
 b2.Location= new System.Drawing.Point(200,250);
 b2.Size= new System.Drawing.Size(50,50);
 b2.Text="Delecte Colors";
 this.Controls.Add(c);
 this.Controls.Add(b1);
 this.Controls.Add(b2);
 b.Click+=new EventHandler(add);
 b1.Click+=new EventHandler(del);
 }
 public void add(Object o,EventArgs e)
 {
 c.Items.Insert(3,"Pink");
 }
 public void del(Object o,EventArgs e)
 {
 c.Items.Remove(c.SelectedItem); // c.Items.RemoveAt(2);
 }
}
public static void Main()
{
 Application.Run(new com());
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

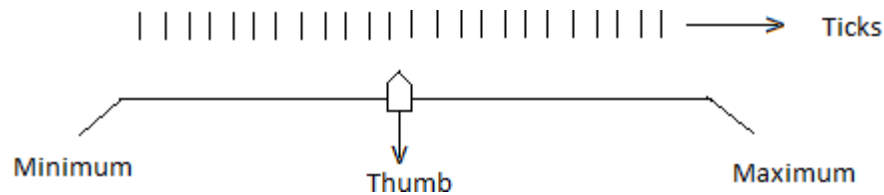
**Sub Name: C# and .NET**

**UNIT III**

**Sub Code: SCSX1008**

**TrackBar**

It allows users to select from a range of values using scrollbar like input mechanism.

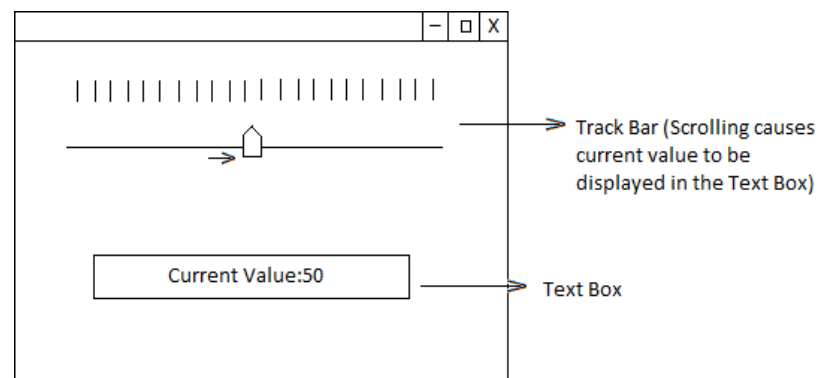


Properties:

1. Maximum : Configure upper bounds of the TrackBar's range.
2. Minimum : Configure lower bounds of the TrackBar's range.
3. Orientation : Orientation for this TrackBar
4. TickFrequency: Indicates how many ticks are drawn
5. Value : Gets or Sets the current location of the track bar.

Event: Scroll

Design the form:



Program:

```
using System.Windows.Forms;
using System.Windows.Forms.Control;
class WinForm:Form
{
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
 TrackBar b1=new TrackBar();
 TextBox t1=new TextBox();
 public WinForm()
 {
 b1.Location=new System.Drawing.Point(50,20);
 b1.tickFrequency=5;
 b1.Minimum=0;
 b1.Maximum=255;
 t1.Location=new System.Drawing.Point(50,50);
 t1.Size=new System.Drawing.Size(50,20);
 this.Controls.Add(b1);
 this.Controls.Add(t1);
 b1.Scroll+=new EventHandler(OnScroll);
 }
 public void OnScroll(object o, EventArgs e)
 {
 Application.Run(new WinForm());
 }
 }
```

**MonthCalendar Control**

This Control is used to select a date (or range of dates) using a Interface.

Properties of MonthCalendar Control:

|                   |                                                                                                                               |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------|
| MaxDate           | Maximum allowable date that can be selected.                                                                                  |
| MaxSelectionCount | The maximum number of days that can be selected.                                                                              |
| MinDate           | The minimum allowable date that can be selected.                                                                              |
| SelectionSort     | Indicate the start date of the selected range of dates                                                                        |
| SelectionEnd      | Indicate the end date of the selected range of dates.                                                                         |
| ShowToday         | Indicates whether the MonthCalendar displays the today date at the bottom of the control, as well as circle the current date. |
| ShowTodayCircle   |                                                                                                                               |

Code for getting the selected date:

```
MonthCalendar m1=new MonthCalendar();
public void Onclick(Object o,EventArgs e)
{
 DateTime d=m1. SelectionStart;
 string str=string.Format("{0}/{1}/{2}", d.Month,d.Day,d. Year);
 MessageBox.Show(str);
}
```

Here the SelectionStart returns the selected date & it will be stored in the variable of DateTime.



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

Properties of DateTime:

|                                         |                                                                |
|-----------------------------------------|----------------------------------------------------------------|
| Day<br>Month<br>Year                    | Extract the day,month & year of the current DateTime type.     |
| Hour<br>Minute<br>Second<br>Millisecond | Extract various time-related details from a DateTime variable. |
| MinValue<br>MaxValue                    | Represent the minimum & maximum DateTime value.                |

**Spin Control(Up/Down Controls)**

- Windows forms provide two widgets that function as spin controls.
- Like the combo box and list box types, these controls allow the user to choose an item from a range of possible selections.
- The difference is that when using DomainUpDown or NumericUpDown control, the information is selected using a small pair of up and down arrows.

Two Types of UpDown Base controls:

1. Domain=> String data
2. Numeric=> Numbers

Properties of UpDown Base:

- Text : Gets or Sets the current text displayed in the spin control.
- TextAlign : Gets or Sets the alignment of the text in the spin control.
- UpDownAlign: Gets or Sets the alignment of the up and down arrows on the spin control.

Properties of DomainUpDown:

- SelectedItem : Returns the selected item.
- SelectedIndex : Returns the index of the selected item.
- Sorted : Configures whether or not the strings should be alphabetized.

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

Event: SelectedItem Changed

```
using System;
using System.Windows.Forms;
using System.Drawing;
class down:Form
{
 DomainUpDown d1 = new DomainUpDown();
 NumericUpDown n1 = new NumericUpDown();
 Label l1 = new Label();
 Label l2 = new Label();
 TextBox t1 = new TextBox();
 TextBox t2 = new TextBox();
 public down()
 {
 d1.Location = new System.Drawing.Point(200,100);
 d1.Sorted = true;
 d1.Wrap=true;
 d1.Items.AddRange(new object[] { "Red","Blue","Green","Pink"});
 d1.SelectedIndex=2;
 n1.Location = new System.Drawing.Point(200,150);
 n1.Maximum = new decimal(5000);
 n1.ThousandsSeparator=true;
 n1.UpDownAlign = LeftRightAlignment.Left;
 l1.Location = new System.Drawing.Point(50,100);
 l1.Size = new System.Drawing.Size(150,20);
 l1.Text = "Domain UpDown Control1";
 l2.Location = new System.Drawing.Point(50,150);
 l2.Size = new System.Drawing.Size(150,20);
 l2.Text="Numeric UpDown Control1";
 t1.Location = new System.Drawing.Point(350,200);
 t1.Size = new System.Drawing.Size(150,20);
 t2.Location = new System.Drawing.Point(550,200);
 t2.Size = new System.Drawing.Size(150,20);
 this.Controls.Add(d1);
 this.Controls.Add(n1);
 this.Controls.Add(l1);
 this.Controls.Add(l2);
 this.Controls.Add(t1);
 this.Controls.Add(t2);
 d1.SelectedItemChanged +=new EventHandler(onselect);
 n1.ValueChanged += new EventHandler(OnClick);
 }
 public void OnClick(object o, EventArgs e)
 {
 t2.Text = string.Format(" number: {0}",n1.Value);
 }
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
public void onselect(object o, EventArgs e)
{
 if(d1.SelectedItem != null)
 {
 string s = d1.SelectedItem.ToString();
 t1.Text=s;
 }
}
public static void Main()
{
 Application.Run(new down());
}
}
```

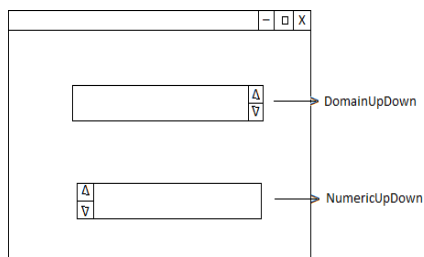
**Properties of NumericUpDown:**

- **Increment** : Sets the numerical value to increment the value in the control.
- **Minimum** : Sets lower limits of the value in the control.
- **Maximum** : Sets upper limits of the value in the control.
- **Value** : Return the current value in the control.

**Event:** Value changed.

Design the form:

**Configure DomainUpDown Control:-**



```
DomainUpDown d1=new DomainUpDown()
d1.Sorted=true;
d1.Items.AddRange(new object [4]{"Red", "Green", "Blue"});
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

**Configure NumericUpDown control:-**

```
NumericUpDown n1= new numericUpDown();
n1.Minimum=1;
n1.Maximum=10;
n1.UpDownAlign=LeftRightAlignment.Left;
```

**Register eventHandler to SelectedItemChanged event of DomainUpDown.d1**

```
d1.SelectedItemChanged+=new EventHandler(OnSelect);
```

**Implementation of OnSelect EventHandler:-**

```
public void onSelect(object o, EventArgs e)
{
 this.Text=d1.Text=d1.Text;
}
```

**Panel**

- GroupBox Control can be used to logically bind a no. of controls (such as RadioButton) to function as collective.
- Panel control is like GroupBox Control.
- Used to group related controls in a logical unit
- Difference is that panel type derives from the scrollableControl class and thus it can support scrollbars which is not possible in GroupBox.

Consider the TrackBars t1, t2, t3.

```
Panel p1=new Panel();
p1.AutoScroll=true//add scrollbar to panel
p1.Controls.Add(t1);//add t1 to panel
p1.Controls.Add(t2);//add t2 to panel
p1.Controls.Add(p1);
this.Controls.Add(p1);//add panel to form
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

**ToolTip**

ToolTip is the small floating point window that display a helpful message when the cursor moreover a given item.

**ToolTip Members:**

- |                   |   |                                                                                                         |
|-------------------|---|---------------------------------------------------------------------------------------------------------|
| 1. Active         | : | Configures if the tooltip is activated or not                                                           |
| 2. AutomaticDelay | : | Gets or Sets the time that passes before the tooltip appears.                                           |
| 3. AutoPopDelay   | : | The period of time that the tooltip remains visible when the cursor is stationary in the tooltip region |
| 4. GetToolTip()   | : | Returns the tooltip text assigned to the specific control.                                              |
| 5. SetToolTip()   | : | Associates a tooltip to a specific control.                                                             |

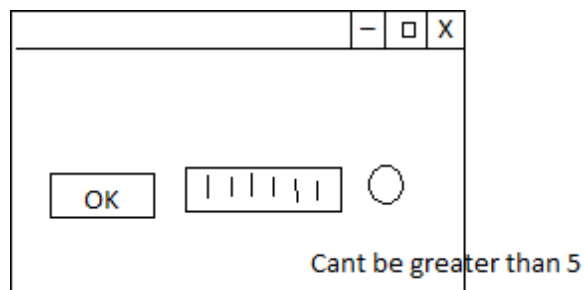
**Code for assigning ToolTip to TextBox Controls:-**

```
TextBox t1=new TetBox();
ToolTip p1=new ToolTip();
p1.Active=True;
p1.SetToolTip(t1,"Enter Text");
```

**Error Provider**

ErrorProvider type can be used to provide a visual of user input error.

Assume you have a form containing a TextBox & Button widget. If the user enters more than five characters in the TextBox, the error information is displayed in the following manner.



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

Properties of Control class:

|                  |                                                                                                  |
|------------------|--------------------------------------------------------------------------------------------------|
| CausesValidation | Indicates whether selecting this control causes validation on the controls requiring validation. |
| Validated        | Occurs when the control is finished performing its validation logic.                             |
| Validating       | Occurs when the control is validating user input.                                                |

// Configure Error Provider

```
ErrorProvider e1= new ErrorProvider();
```

```
e1.BlinkStyle= System.Windows.Forms.ErrorBlink;
```

```
e1.BlinkStyle=500;
```

BlinkStyle: It can take any of the values of the ErrorBlinkStyle enumeration.

ErrorBlinkStyle Properties:

|                       |                                                                                                                           |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------|
| AlwaysBlink           | Causes the error icon to blink when the error is first displayed.                                                         |
| BlinkIfDifferentError | Causes the error icon to blink only if the error icon is already displayed but a new error string is set for the control. |
| NeverBlink            | Indicates the error icon never blinks.                                                                                    |

Bind error to the TextBox within the scope of its validating eventhandler.

Code:

```
public void ExInput_validating(object o,EventArgs e)
{
 If(t1.Text.Length>5)
 {
 e1.SetError(t1,"cant be greater than 5!");
 }
else
 e1.SetError(t1,"");
}
}
```

### **DialogBox**

- It is used to get input from the user.
- DialogBox is considered as a stylized form.

Dialogue boxes are typically configured non-sizable. Therefore we set BorderStyle.

Property to FormBorderStyle.FixedDialog

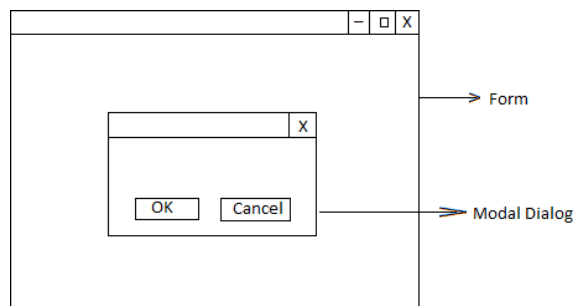
Set the Control Box, Minimize Box and Maximize properties to false.

Two types:

1. Modal Dialog
2. Modeless Dialog

### **Modal Dialog:**

Definition: The form cannot receive the focus until the dialogbox is closed.



Use the ShowDialog() for displaying the Modal Dialog.

### **Code for Modal Dialog:**

```
public void Onclick(Object o,EventArgs e)
{
 someform s1=new someform();
 f1.BorderStyle=FormBorderStyle.FixedDialog;
 f1.ControlBox=false;
 f1.MinimizeBox=false;
 f1.MaximizeBox=false;
 f1.ShowDialog(this); // To display Modal DialogBox
}
```

**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

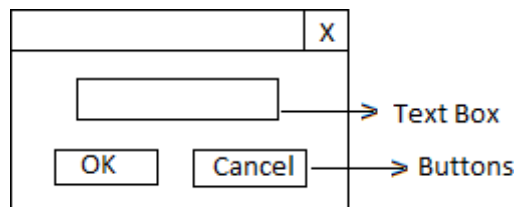
**UNIT III**

**Sub Code:SCSX1008**

**Code for Modeless Dialog:**

```
public void OnClick(object o, EventArgs e)
{
 • ---
 • ---
 fl.Show(this) // To display Modeless Dialog
}
```

**DialogBox Creation:**



```
public class SampleDialog:Form
{
 Button btnOk=new Button();
 Button btnCancel=new Button();
 TextBox t1=new TextBox();
 public SampleDialog()
 {
 btnOk.Location=new System.Drawing.Point(20,70);
 btnOk.Size=new SytemDrawing.Size(30,20);
 btnOk.Text="OK";
 btnOk.DialogResult=DialogResult.Ok;
 btnCancel.Location=new System.Drawing.Point(60,70);
 btnCancel.Size=new System.Drawing.Size(30,20);
 }
}
```



**SATHYABAMA UNIVERSITY**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**  
**COURSE MATERIAL**

**Sub Name:C# and .NET**

**UNIT III**

**Sub Code:SCSX1008**

```
btnCancel.Text="Cancel";
btnCancel.DialogResult=DialogResult.Cancel;
t1.Location=new System.Drawing.Point(20,30);
t1.size=new System.Drawing.Size(50,20);
this.Controls.Add(btnOk);
this.Controls.Add(btnCancel);
this.Controls.Add(t1);
this.ControlBox=false;
this.MaximizeBox=false;
this.MinimizeBox=false;
this.BorderStyle=FormBorderStyle.FixedDialog;
}
}
```

*Access this DialogResult:*

Consider the eventhandler of some other class (Form)

```
public void Onclick(Object o,EventArgs e)
{
 SampleDialog s1=new SampleDialog();
 s1.Show(this);
 if(s1.DialogResult==DialogResult.Ok)
 {
 // Write code for Ok button
 }
}
```

## UNIT IV

Input and output – Introduction to System. IO .namespace

File and folder operations

Stream class

Introduction to ADO .NET

Building data table

Data view

Data set

Data relations

ADO.NET managed providers – OleDb managed provider

SQLProvider.

### 5.1 Introduction to System.IO namespace

Used to manipulate the physical directories and files.

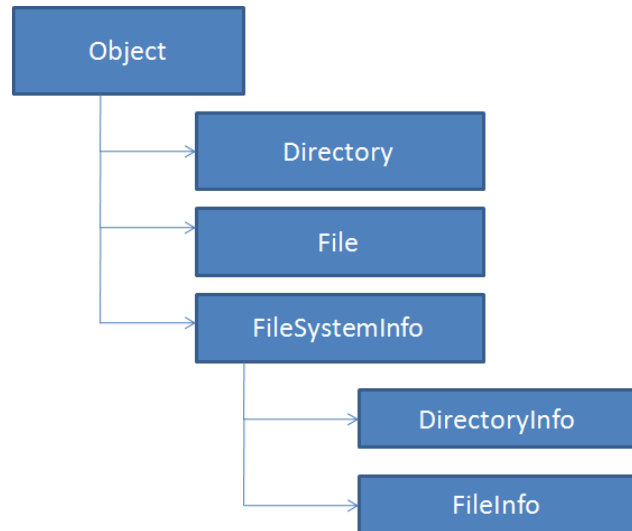
To read data from and write data to string buffers.

#### Members of System.IO namespace

| IO Type                                        | meaning                                                                                                                                                                                                                                 |
|------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Directory<br>DirectoryInfo<br>File<br>FileInfo | Used to manipulate the properties for a given directory or file.<br>The Directory and File types expose their functionality as static methods.<br>The DirectoryInfo and FileInfo types expose similar functionality as Object variable. |
| Path                                           | Contains file directory path information.                                                                                                                                                                                               |
| StreamWriter<br>StreamReader                   | Used to store text information to a file. Do not support for random access.                                                                                                                                                             |
| StringWriter<br>StringReader                   | Same as StreamReader/Writer. but storage is a string buffer rather than a file.                                                                                                                                                         |
| FileStream                                     | For random file access.                                                                                                                                                                                                                 |
| BinaryReader<br>BinaryWriter                   | Used to store and retrieve primitive data types(Integer,Boolean,strings) as a binary value.                                                                                                                                             |

## 5.2 File and Folder Operations

### File and Directory types



### FileSystemInfo

| FileSystemInfo Property | Meaning                                                 |
|-------------------------|---------------------------------------------------------|
| Attributes              | Gets or sets the attributes associated to current file. |
| CreationTime            | Gets or sets the creation for the file or directory.    |
| Exists                  | Used to determine if a given file or directory exists.  |
| Extension               | Used to retrieve a file extension.                      |
| FullName                | Gets the full path of the directory or file.            |

### DirectoryInfo

| Members                          | Meaning                                                 |
|----------------------------------|---------------------------------------------------------|
| Create()<br>CreateSubdirectory() | Create a directory or subdirectory.                     |
| Delete()                         | Deletes a directory all its contents.                   |
| GetDirectories()                 | Returns an array of strings to list the subdirectories. |
| GetFiles()                       | Gets the files in the specified directory.              |
| MoveTo()                         | Moves a directory and its contents to a new path.       |

### Program to display details of directory

Class direct

```
{
 public static void Main()
 {
 DirectoryInfo dir=new DirectoryInfo("c:\\sharp");
 Console.WriteLine("Fullname" + dir.FullName);
 Console.WriteLine("Creation" + dir.CreationTime);
 Console.WriteLine("Attributes" + dir.Attributes.ToString());
 }
}
```

### FileInfo

Used to retrieve the details of existing files.

Used to create, copy, move and delete the files.

### **Members of FileInfo**

| Members | Meaning |
|---------|---------|
|---------|---------|

|             |                                        |
|-------------|----------------------------------------|
| Create()    | Creates the new file.                  |
| CopyTo()    | Copies an existing file to a new file. |
| MoveTo()    | Moves the file to a new location.      |
| Delete()    | Remove the file from directory.        |
| Open()      | Open the already existing file.        |
| OpenRead()  | Creates a read-only filestream.        |
| OpenWrite() | Creates a read/write filestream.       |

#### **Program to display details of the given file**

```
using System;
using System.IO;
class dir
{
 public static void Main()
 {
 FileInfo f=new FileInfo("c://c#/sample1.txt");
 FileStream fs=f.Create();
 Console.WriteLine(f.CreationTime + "\n"+ f.FullName + "\n" +
 f.Attributes.ToString());
 fs.Close();
 f.Delete();
 }
}
```

#### **FileInfo.Open()-Opens File with various read and write previlages**

Syntax: FileStream f1= FileInfo.Open(FileMode.Create,FileAccess.Read,FileShare.Read);

#### **FileMode Attribute Values**

| FileMode | Meaning                                                                                               |
|----------|-------------------------------------------------------------------------------------------------------|
| Append   | Open the already existing file and write at the end of the file. if it not exist create the new file. |

|              |                                                                          |
|--------------|--------------------------------------------------------------------------|
| Create       | Create the new file. If the file already exists, it is overwritten.      |
| CreateNew    | Create a new file. If the file already exists, an IOException is thrown. |
| Open         | Open an existing file.                                                   |
| OpenOrCreate | Open an existing file; otherwise creates new file.                       |
| FileAccess   | Meaning                                                                  |

#### FileAccess Attribute Values

| FileAccess | Meaning                            |
|------------|------------------------------------|
| Read       | Read-only access to the file.      |
| ReadWrite  | Read and write access to the file. |
| Write      | Write access to the file.          |

#### FileShare Attribute Values

| FileShare | Meaning                                                       |
|-----------|---------------------------------------------------------------|
| None      | Declines sharing of the current file.                         |
| Read      | Allows subsequent opening of the for reading.                 |
| ReadWrite | Allows subsequent opening of the file for reading or writing. |
| Write     | Allows subsequent opening of the file for writing.            |

## Stream

### Members of class Stream

| Stream Member          | Meaning                                                                            |
|------------------------|------------------------------------------------------------------------------------|
| Close()                | Closes the current stream.                                                         |
| Length                 | Returns the length of the stream, in bytes.                                        |
| Position               | Determines the position in the current stream.                                     |
| Read()<br>ReadByte()   | Reads a sequence of bytes and advances the current position.                       |
| Seek()                 | Sets the position in the current stream.                                           |
| Write()<br>WriteByte() | Write a sequence of bytes to the current stream and advances the current position. |

## **ByteStreams**

### **a) FileStreams – Example**

```
using System;
using System.IO;
class stream
{
 public static void Main()
 {
 FileStream fs=new FileStream("test.txt",FileMode.OpenOrCreate,
 FileAccess.ReadWrite);

 for(int i=0; i<256; i++)
 fs.WriteByte((byte)i);
 fs.Position=0;
 for(int i=0; i<256; i++)
 Console.WriteLine(fs.ReadByte());
 fs.Close();
 }
}
```

### **b) Memory Streams – Example**

```
using System;
```

```

using System.IO;
class stream
{
 public static void Main()
 {
 MemoryStream ms=new Memorytream();
 ms.Capacity=256;
 for(int i=0; i<256; i++)
 ms.WriteByte((byte)i);
 ms.Position=0;
 for(int i=0; i<256; i++)
 Console.WriteLine(ms.ReadByte());
 ms.Close();
 }
}

```

### **c)Buffered Stream – Example**

```

using System;
using System.IO;
class stream
{
 public static void Main()
 {
 FileStream fs=new FileStream("test.txt",FileMode.OpenOrCreate,FileAccess.ReadWrite);
 BufferedStream bs=new BufferedStream();
 Byte[] b={0x55,0x66,0x43};
 Bs.Write(b,0,b.Length)
 bs.Close();
 }
}

```

## **Character Streams**

### **Members of TextWriter**

| TextWriter Member | Meaning                                                        |
|-------------------|----------------------------------------------------------------|
| Close()           | Closes the writer, the buffer is automatically flushed.        |
| Write()           | Writes a line to the text stream, without a new line constant. |



|             |                                                             |
|-------------|-------------------------------------------------------------|
| WriteLine() | Writes a line to the text stream, with a new line constant. |
| NewLine     | Used to make a newline.                                     |
| Flush()     | Clears all buffers for the current writer.                  |

#### Members of TextReader

| TextReader Member | Meaning                                                                                   |
|-------------------|-------------------------------------------------------------------------------------------|
| Read()            | Reads data from an input stream.                                                          |
| ReadBlock()       | Reads a max. of count characters from the current stream and writes the data to a buffer. |
| ReadLine()        | Reads a line of characters from the current stream and returns data as string.            |
| ReadToEnd()       | Reads all character from the current position till end and returns them as one string.    |
| Peek()            | Returns the next character without changing the position of the reader.                   |

#### Stream Reader and Writer

Used to read from or write Character based data to file(e.g. string).

Object

- TextReader
  - StreamReader
  - StringReader
- TextWriter
  - StreamWriter
  - StringWriter

#### Example-StreamWriter

Program for Write the text content in to the file sample.txt

Class wstream

```
{
 public static void Main()
 {
```

```

 FileInfo f=new FileInfo("sample.txt");
 StreamWriter sw=f.CreateText();
 sw.WriteLine("Good Morning");
 sw.WriteLine("Have a nice day");
 sw.Close();
 }
}

```

### StreamReader

#### Program for reading the data from sample.txt

Class sread

```

{
 public static void Main()
 {
 // create a StreamReader
 StreamReader sr=File.OpenText("sample.txt");
 string input;
 while((input=sr.ReadLine())!=null)
 Console.WriteLine(input);
 sr.Close();
 }
}

```

### StringWriter and StringReader

Used to read from or writeCharacter based data to Buffer(e.g. string).

#### StringWriter - Example

Class strwrite

```

{
 public static void Main()
 {
 StringWriter sw=new StringWriter();
 sw.WriteLine("good morning");
 sw.WriteLine("Have a nice day");
 sw.Close();
 Console.WriteLine("contents" + sw.ToString());
 }
}

```

### StringReader

#### Code to read the content from StringWriter:

```

StringReader sr=new StringReader(sw.ToString());
String input;
While((input=sr.ReadLine())!=null)
 Console.WriteLine(input);
Sr.Close();

```

### **5.3.3.BinaryReader and BinaryWriter**

Used to read and write discrete data types to an underlying stream.

#### **Members of Binary Writer**

Base Stream  
Close()  
Flush()  
Seek()  
Write()

#### **Members of BinaryReader**

BaseStream  
Close()  
PeekChar()  
Read()  
ReadXXX()

#### **Example Program**

```
Using System;
Using System.IO.*;
Class sampleBINARY
{
 FileStream fs=new FileStream("temp.dat",FileMode.OpenOrCreate,FileAccess.Read);
 BinaryWriter w=new BinaryWriter(fs);
 w.Write(10);
 w.Write(5.78)
 w.Write('a');
 w.BaseStream.Position=0;
 BinaryReader r=new BinaryReader(fs);
 While(r.PeekChar!=-1)
 {
 Console.WrieLine(r.ReadByte());
 }
}
```

#### **Introduction to ADO.NET**

ADO .NET is a collection of classes, interfaces, structures, and enumerated types that manage data access from relational data stores within the .NET Framework

#### **Evolution of ADO.NET**

The first data access model, DAO (data access model) was created for local databases.

Next came RDO (Remote Data Object) and ADO (Active Data Object) which were designed for Client Server architectures.

With ADO, all the data is contained in a recordset object which had problems when implemented on the network and penetrating firewalls.

ADO was a connected data access, which means that when a connection to the database is established the connection remains open until the application is closed.

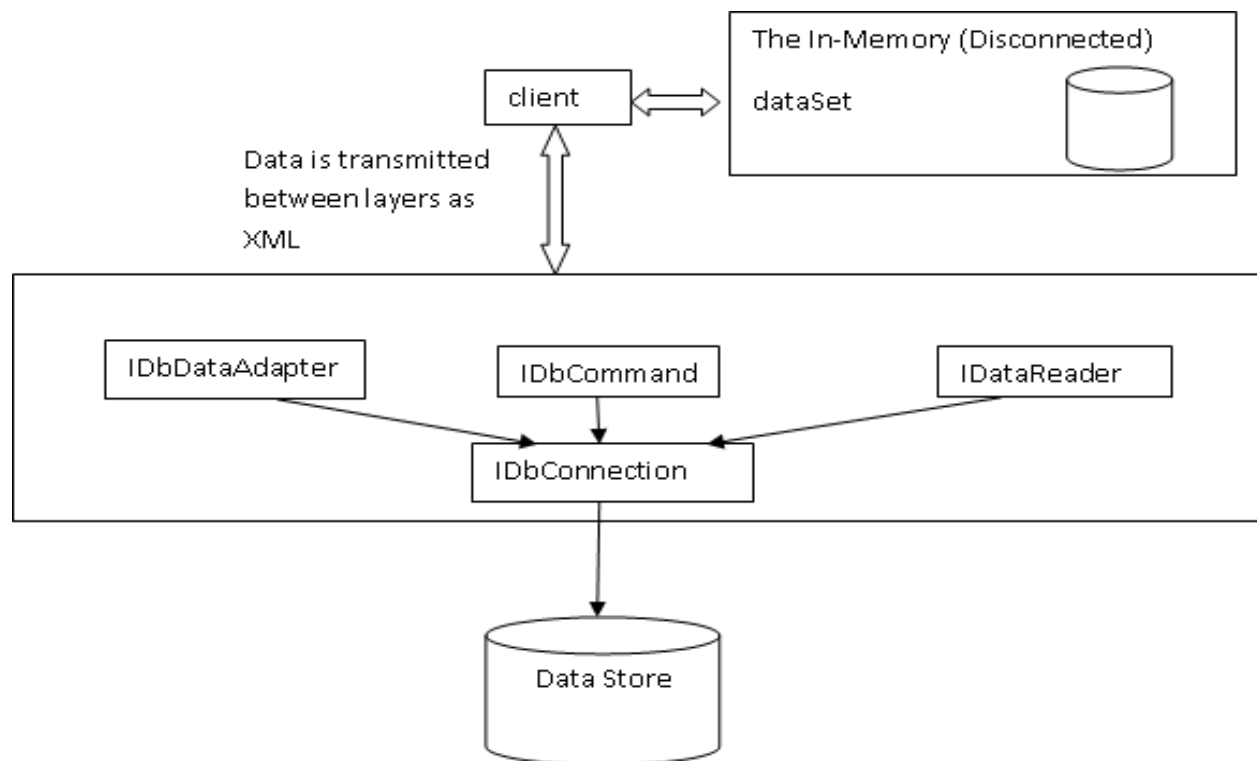
ADO .NET addresses the above mentioned problems by maintaining a disconnected database access model which means, when an application interacts with the database, the connection is opened to serve the request of the application and is closed as soon as the request is completed. If a database is Updated, the connection is opened long enough to complete the Update operation and is closed.

By keeping connections open for only a minimum period of time, ADO .NET conserves system resources and provides maximum security for databases and also has less impact on system performance.

#### **ADO versus ADO .NET**

| <b>Feature</b>                                | <b>ADO</b>                                                    | <b>ADO.NET</b>                                                        |
|-----------------------------------------------|---------------------------------------------------------------|-----------------------------------------------------------------------|
| <b>Primary Aim</b>                            | <b>Client/server coupled</b>                                  | <b>Disconnected collection of data from data server</b>               |
| <b>Form of data in memory</b>                 | <b>Uses RECORDSET object (contains one table)</b>             | <b>Uses DATASET object (contains one or more DATATABLE objects)</b>   |
| <b>Disconnected access</b>                    | <b>Uses CONNECTION object and RECORDSET object with OLEDB</b> | <b>Uses DATASETCOMMAND object with OLEDB</b>                          |
| <b>Disconnected access across multi-tiers</b> | <b>Uses COM to marshal RECORDSET</b>                          | <b>Transfers DATASET object via XML. No data conversions required</b> |

#### **ADO.NET Architecture**



## Components

Data Access in ADO.NET relies on two components:

- DataSet
- DataProvider

DataSet

- The dataset is a disconnected, in-memory representation of data.
- It is Local copy. The data can be Manipulated and updated independent of database.

Disconnected, cached, scrollable data

DataProvider

- The Data Provider is responsible for providing and maintaining the connection to the database.
- There are two DataProviders:
  - SQL Data Provider (specifically for SQL Server7.0)
  - OleDb DataProvider (other databases like access,oracle)

Each DataProvider consists of the following component classes:

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

### **ADO.NET Namespaces**

|                          |                                                                                            |
|--------------------------|--------------------------------------------------------------------------------------------|
| System.Data              | This namespace defines types that represent tables, rows, columns, constraints & DataSets. |
| System.Data.Common       | This namespace contains types shared between data providers.                               |
| System.Data.OleDb        | This namespace defines types that allow you to connect to an OLE.DB                        |
| System.Data.Odbc         | This namespace defines types that constitute the ODBC data provider.                       |
| System.Data.OracleClient | This namespace defines types that constitute the Oracle data provider.                     |
| System.Data.SqlClient    | This namespace defines types that constitute the SQL data provider.                        |

### **Building Data Table**

#### **Types of System. Data**

| Type                                   | Meaning                                                                                                                                         |
|----------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| DataColumnCollection<br>DataColumn     | DataColumnCollection is used to represent all of the columns used by a given DataTable. DataColumn represents a specific column in a DataTable. |
| DataRowCollection<br>DataRow           | These types represent a collection of rows for a DataTable and a specific row of data in a DataTable.                                           |
| DataSet                                | Represents an in-memory cache of data that may consist of multiple related DataTables.                                                          |
| DataRelationCollection<br>DataRelation | This collection represents all relationships between the tables in a DataSet                                                                    |
| DataTableCollection<br>DataTable       | This collection represents all of the tables for a particular DataSet.                                                                          |

#### **DataColumn**

1. The DataColumn type represents a single column maintained by a DataTable

2. Eg: Assume you have a table named Employees with three columns (EmpID, FirstName and LastName)

Properties of DataColumn:

| Column Name                                             | Meaning                                                                                 |
|---------------------------------------------------------|-----------------------------------------------------------------------------------------|
| DataType                                                | Defines the datatype (Boolean, string, float & so on) stored in the column.             |
| AllowDBNull                                             | Boolean value that indicates whether the column may contain null values.                |
| Expression                                              | An expression defining how the value of a column is calculated.                         |
| Caption                                                 | The caption to be displayed for this column.                                            |
| AutoIncrement<br>AutoIncrementSeed<br>AutoIncrementStep | These properties are used to configure the auto increment behaviour for a given column. |
| ReadOnly                                                | Determines if this column can be modified once a row has been added to the table.       |
| Table                                                   | Gets the DataTable that contains this DataColumn.                                       |

Enabling Auto.Incrementing Fields:-

AutoIncrementing columns are used to ensure that when a new row is added to a given table the value of this column is assigned automatically based on the current step of the incrementation.

Seed value is used to mark the starting value of the column, where step value identifies the number to add to the seed when incremented

eg:-

```
DataTable t1=new DataTable("Student");
DataColumn C1=new DataColumn();
C1.ColumnName ="ID"
C1.ColumnName=System.Type.GetType("System.Int32");
C1.AutoIncrement=true;
C1.AutoIncrementseed=1;
C1AutoIncrementStep=1;
DataColumn C2=new DataColumn();
C2.ColumnName="Name";
C2.DataType=Type.GetType("System.String");
C2.ReadOnly=True;
t1.columns.Add(C1);
```

```

t1.columns.Add(C2);

DataColumn C3=new DataColumn();
C3.ColumnName="Dept";
C3.DataType=Type.GetType("System.String");
t1.columns.Add(C1);
t1.columns.Add(C2);
t1.columns.Add(C3);

```

### **Data Row**

- \*Data Row types represents the actual data in the table.
- \*cannot create direct instance of this type, but obtain reference from a given Data Table.
- \*Use the method New Row() of Data Table

### **Members of DataRow Type:**

ItemArray : This property gets or sets all of the values for this row using an array of objects.

Table : Obtaining the reference of the table containing this row.

AcceptChanges: Commit all the changes made to this row.

RejectChanges: Reject all the changes made to this row.

Delete() : Marks this row to be removed.

IsNull() : Get the value indicating whether the specified column contains null

### **How to insert new row in the above table:**

```

DataTable t1=new DataTable("Student");
//.....
.....
Creat two columns "Names" & ID.
DataRow r1=t1.NewRow();//Creat Row
r1["name"]="aaa";//insert datas
r1["Dept"]="IT";
t1.Rows.Add(r1);//Add row to the table.
DataRow r2=t1.NewRow();//Creat Row
r2["name"]="bbb";//insert datas
r2["Dept"]="IT";
t1.Rows.Add(r2);//Add row to the table.
DataRow r3=t1.NewRow();//Creat Row
r3["name"]="aaa";//insert datas
r3["Dept"]="ECE";
t1.Rows.Add(r3);//Add row to the table.

```

### **Data Table**

Data table is an in-memory representation of a tabular block of data.



Members of data table:

Columns : Returns the collection of columns that belong to this table.  
Constraints : Gets the collection of constraints maintained by the table.  
DefaultView : Gets a customized view of the table.  
MaximumCapacity: Gets or Sets the initial no. of rows in this table.  
PrimaryKey : Gets or Sets an array of columns that function as primary keys for the table.  
Rows : It returns the collection of rows that belong to this table.  
TableName : Gets or Sets the name of the table.

Now the structure of Table student:

| ID | Name | Dept |
|----|------|------|
| 1  | aaa  | IT   |
| 2  | bbb  | IT   |
| 3  | aaa  | ECE  |

Building Complete DataTable:

### **DataTable**

Create the following Data Table

| ID | Name |
|----|------|
| 1  | aaa  |
| 2  | bbb  |
| 3  | aaa  |
| 4  | bbb  |

Program to create above table

Using System.Data;

Using System;

Class sample

{

Public static void Main()

{

DataTable t1=new DataTable("Student");

//Create Data Column "ID" and add it to the data table.

```
DataColumn c1=new DataColumn();
c1.ColumnName="ID";
c1.DataType=Type.GetType("System.Int32");
//Set the Auto-Increment behaviour
c1.AutoIncrement=true;
c1.AutoIncrementSeed=1;
c1.AutoIncrementStep=1;
//Add this column
t1.Columns.Add(c1);
```

//Create DataColumn "Name" and add it to the table.

```
DataColumn c2=new DataColumn();
c2.ColumnName= "Name";
c2.DataType=Type.GetType("System.String");
//add this column
t1.Columns.Add(c2);
```

//Create New Row and insert datas.

```
DataRow r1=t1.NewRow();
r1["Name"] = "aaa";
//Value of ID field set automatically
//Add row to DataTable
t1.Rows.Add(r1);
DataRow r2=t1.NewRow();
r2["Name"] = "bbb";
//Value of ID field set automatically
//Add row to DataTable
t1.Rows.Add(r2);
DataRow r3=t1.NewRow();
r3["Name"] = "aaa";
//Value of ID field set automatically
//Add row to DataTable
t1.Rows.Add(r3);
//Create some more rows
t1.AcceptChanges();
}
}
```

#### Manipulating a DataTable

**A data table can be manipulated by three operation**

**Selection**

**Updation**

**Deletion**

## SELECTION OF SPECIFIED ROWS

\*use the select() method

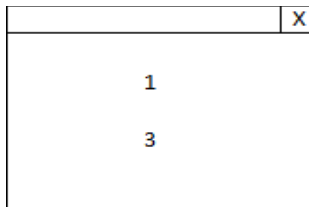
\*parameter sent to select() is a string that contains some conditional operation

**select rows with 'name=aaa' from the table 2.1 (student) and extract ID for that students.**

code:-

```
string s1="name='aaa' ";
DataRow[] r1=t1.select(s1);
t1=>table reference
if(r1.Length==0)
 MessageBox.Show("norecords");
else
{
 string str=null;
 for(int i=0;i<r1.Length;i++)
 {
 DataRow t=r1[i];
 str+=t["ID"]+"\n";
 }
 MessageBox.Show(str);
}
```

o\p



|   |
|---|
| 1 |
| 3 |

## UpDation:

**Searches the DataTable student for all rows where name is equal to aaa.Once you identify these items, change the name from "aaa" to "ddd"**

code:-

```
string s1="Name='aaa' ";
DataRow[] r1=t1.Select(s1);
for(int=0;i<r1.Length;i++)
{
 DataRow t=r1[i];
 t["name"]="ddd";
 r1[i]=t;
}
```

### **Deletion of Rows**

- \* use the Delete() method.
- \* specify the index representing the row to remove.

Eg..

```
// delete the first row of the student table
t1.Rows[0].Delete();
t1.AcceptChanges();
```

### **DataView**

- \*View object is a stylized representation of a table.
- \*DataView type allows you to programmatically extract a subset of data from a DataTable.
- \*you can hold multiple views of same Table.

eg:-

- \*create two DataViews.
- View1 -> rows with name="aaa";
- View2 -> rows with name="bbb";
- \*Bind those views to Data Grid (Container)

### **Source Code:**

```
Dataview v1,v2;
Data Grid g1,g2;
v1=new DataVeiw(t1);
v2=new DataView(t1);
//t1 -> reference for student DataTable.
v1.RowFilter="name='aaa'";
v2.RowFilter="name='bbb'";
//RowFilter->propertyn gets or sets the expression used to filter which rows are viewed in DataView.
g1=new DataGrid();
g2=new DataGrid();
g1.Data Source=v1;
g2.Data Source=v2;
//g1->Contains View v1
//g2->Contains View v2
```

### **Members of DataView Type:**

sort                      gets or sets the sort column or columns and sort order for the table.

Delete()                Deletes a row at a sepcified index.

RowFilter               already explained.

AddNew()               Adds a new row to the DataView.

### **DataSet:**

DataSet is an in-memory representation of any number of tables as well as any relationships between three tables and any constraints

### Members of DataSet:

DataSetName: Represents the friendly name of this DataSet.

Relations: Gets the collection of relations that link tables and allows navigation from parent DataTables to child DataTables.

Tables : Provides access to the collection tables maintained by the DataSet.

Clear(): Completely clear the DataSet data.

Clone(): Clones the structure of the DataSet

GetChild Relations(): Returns the collection of child related that belong to the specified table.

### Add Tables to DataSet:

```
Code: Data Set s1=new Data Set("Inventory");
 DataTable t1=new DataTable("customer");
 DataTable t2=new DataTable("orders");
//Add Tables to DataSet
 s1.Tables.Add(t1);
 s1.Tables.Add(t2);
```

### Access Rows of the specified table:

```
s1.Tables["orders"].Rows[1];
//access 1st row of orders table of DataSet s1.
```

### Data Relation:

\*Once a DataSet has been populated with a number of tables, you can programmatically model their parent/child relationships.

\*For a relationship to be established each table must have an identically named column of the same data type

DataSet s1:student,details

ID=>identically named column of same datatype

### Creation of Relation:

```
Data Relation dr=new Data Relation("student
Details",s1.Tables["student"].columns["ID"],s1.Tables["details"].columns["ID"]);
s1.Relations.Add(dr);
```

### Properties of DataRelation:

Obtains information about the child table in the relationship

Child Columns

ChildKeyConstraint

Child Table

obtain information about the parent table in the relationship

Parent Columns

ParentKeyConstraint

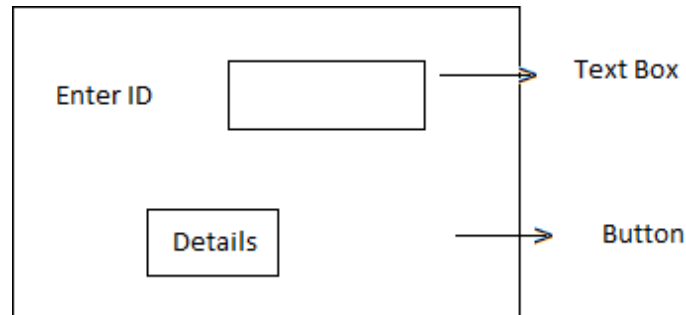
ParentTable

Relation Name                      Gets or sets the name the name of the relation

#### Navigation between Related Tables:-

By means of samples you will have a Data relation that allows you to

#### Design the form:-



While clicking button "Details" get name from the students table and Languages know from the Details Data table and display that using the message box.

#### sample code for eventhandler:-

```
public void b1-click(object sender,Event Args e)
{
string str=" ";
DataRow r1=null;
DataRow[] r2=null;
int c1=int .parse(t1.Text);
r1 =s1.Tables["student"].Rows[c1];
str=r1["name"];
r2=r1.GetChildRows(s1.Relations["studentDetails"])
foreach(Data Row d in r2)
str+=d["lang"];
Message Box.Show(str);
}
```

#### **Data Provider**

\*Used to access data from Data store.

#### \*Major Types:

\*OleDb DataProvider

\*SQL Data Provider

#### \*OleDb DataProvider:-

Used to access data located in any data store that supports the classic OLEDB protocol

NameSpace:-System.Data.OleDb;

\*SQL Data provider:

used to access data from SQL server data stores.

NameSpace:-System.Data.SqlClient

ADO.NET Providers:-

- |                           |                               |
|---------------------------|-------------------------------|
| 1.Microsoft.JET.OLEDB.4.0 | connect to an access database |
| 2.MSDAORA                 | connect to an Oracle server   |
| 3.SQLOLEDB                | connect to the SQL server.    |

**.1 OleDb Data Provider:**

Members of System.Data.OleDb:-

OleDbCommand Represents a SQL query command to be made to a data source  
OleDb Connection Represents an open connection to a data source  
OleDb DataReader provides a way of reading a forward-only stream of data records from a data source  
OleDb DataAdapter represents a set of data commands and a data connection used to fill and update the contents of a Data Set.

Create connections:-

```
OleDb Connection cn=new OleDb Connection();
cn.Connection String="Provider=SQLOLEDB.1;"+"User ID=sa;pwd;"+"@"Data Source=c: \sample;"
cn.Open();
cn.Close();
```

Members of the OleDb Connection:-

connection string: Gets or sets the string used to open a session with a data store.  
Database : gets the name of the database maintained by the connection object.  
DataSource : Gets the location of the data base maintained by the connection object.  
Open() : opens a data base connection  
Provider : Gets the name of the provider maintained by the connection object.  
Close() : closes the connection to the datasource.

OleDb Command:-

By using OleDb Command class you can submit SQL queries to the database.

eg:-

```
//specify SQL command and connection as
//Constructor parameters.
String str="select*from student where name='aaa'";
OleDbCommand co=new OleDbCommand(str,cn);
//specifySQL command and connection via properties.
string str1="select *from student where name='aaa'";
OleDb Command co=new OleDb Command();
co.connection=cn;
co.commandText=str1
```

#### Members of OleDb Command Type:-

|                   |                                                                                                              |
|-------------------|--------------------------------------------------------------------------------------------------------------|
| Command text      | Gets or sets the SQL command text to run against the data source                                             |
| Connection        | Gets or sets the OleDb connection                                                                            |
| ExecuteReader()   | Returns an instance of an OleDb DataReader which provides forward only ,read only access to underlying data. |
| ExecuteNonQuery() | This method issues the command text to the data store without returning on OleDbDataReader type              |

#### Code to Access data from data store

```
Using System.Data;
Using System;
Class Data
{
 Public static void Main()
 {
 //Make connections
 OleDbConnections cn=new OleDbConnection();
 Cn.ConnectionString="Provider=SQLOLEDB.1";+"UserID
 =pa;Pwd"; +"DataSource=c:\Sample";

 Cn.Open();
 //create SQL Command
 String str="Select * from student";
 OleDbCommand co=new OleDbCommand(str,cn);
 //obtain DataReader via ExecuteReader()
 OleDbDataReader re;
 Re=co.ExecuteReader();
 //obtain the records through DataReader
 while(re.Read())
 {
 Console.WriteLine(re["ID"]);
 }
 }
}
```

#### **5.9.2 OleDbDataReader**

DataReaders are useful only when submitting SQL selection statements to underlying data store.

Program:

using System;



```

using System.Data.OleDb;
class Data
{
 //close the DataReader and connection
 re.close();
 cn.close();
}
}

```

### Inserting ,updatingand deleting records using OleDbCommand

#### Insertion:

```

String str="Insert into Inventory"+"(name,ID)values"+"('aaa','123');
OleDbCommand co=new OleDbCommand(str,cn);
Co.ExecuteNonQuery();

```

#### Updation:

```

String str="Update student set name="bbb" where name='aaa'";
OleDbCommand co=new OleDbCommand(str,cn);
Co.ExecuteNonQuery();

```

#### Deletion:

```

String str="Delete from student where name='aaa'";
OleDbCommand co=new OleDbCommand(str,cn);
Co.ExecuteNonQuery();

```

### Example

#### **SqlProvider**

**Used to access data from SQL database.**

Syntax for creating Sql Provider is similar to OleDb provider.**Instead of OleDb write Sql**

Eg: OleDbConnection->SqlConnection  
 OleDbCommand ->SqlCommand

#### **Example**

```

using System;
using System.Data;
using System.Data.SqlClient;

class MainClass
{
 static void Main(string[] args)
 {

```

```

string connString = "server=(local)\\SQLEXPRESS;database=MyDatabase;Integrated Security=SSPI;";

string sql = @"select * from employee";

SqlConnection conn = null;
SqlDataReader reader = null;

try
{
 conn = new SqlConnection(connString);
 conn.Open();

 SqlCommand cmd = new SqlCommand(sql, conn);
 reader = cmd.ExecuteReader();

 Console.WriteLine("Querying database {0} with query {1}\n", conn.Database, cmd.CommandText);

 while(reader.Read()) {
 Console.WriteLine("{0} | {1}", reader["FirstName"].ToString().PadLeft(10), reader[1].ToString().PadLeft(10));
 }
} catch (Exception e)
{
 Console.WriteLine("Error: " + e);
}
finally
{
 reader.Close();
 conn.Close();
}
}

```

## UNIT V

**Web development and ASP.NET**  
**Web applications and web servers**  
**HTML form development**  
**Client side scripting**  
**GET and POST**  
**ASP.NET application**  
**ASP.NET namespaces**  
**Creating sample C# web Applications.**  
**Understanding Web Security**  
**Windows authentication**  
**Forms authentication**  
**Web services**  
**Web service clients**  
**The CityView application.**

### **Web development and ASP.NET**

ASP.NET is a development framework for building web pages and web sites with HTML, CSS, JavaScript and server scripting.

ASP.NET supports three different development models: Web Pages, MVC (Model View Controller), and Web Forms:

.NET framework to be the most robust and flexible technology available for developing web applications. It is quick to develop, runs very fast and can also be used with other technologies which makes it very powerful.

### **Web applications :**

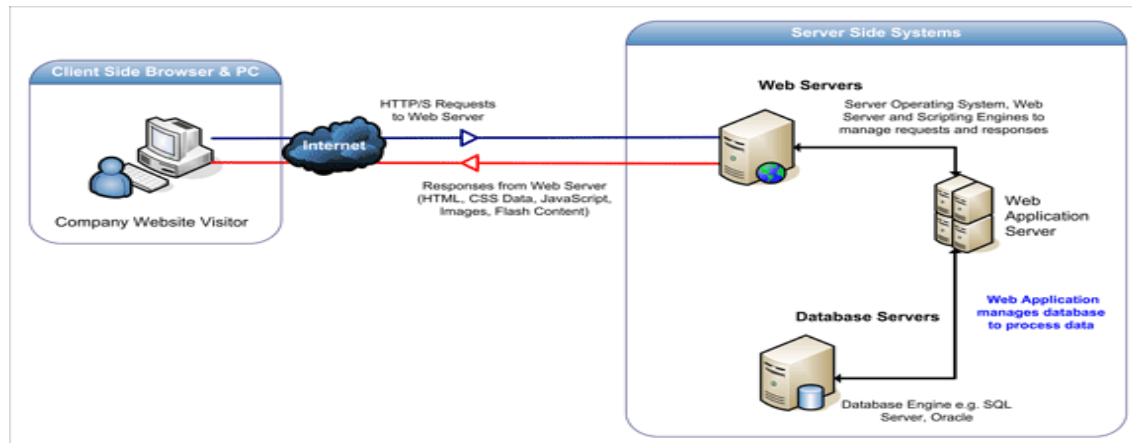
A web application or web app is any application software that runs in a web browser or is created in a browser-supported programming language (such as the combination of JavaScript, HTML and CSS) and relies on a common web browser to render the application.

### **What is a Client?**

The 'client' is used in client-server environment to refer to the program the person uses to run the application. A client-server environment is one in which multiple computers share information such as entering information into a database. The 'client' is the application used to enter the information, and the 'server' is the application used to store the information.

Web applications commonly use a combination of server-side script (ASP, PHP, etc) and client-side script (HTML, Javascript, etc.) to develop the application. The client-side script deals with the presentation of the information while the server-side script deals with all the hard stuff like storing and retrieving the information.

## How web Application Work :



### Web Pages : Single Pages Model

- Simplest ASP.NET model.
- Similar to PHP and classic ASP.
- Built-in templates and helpers for database, video, graphics, social media and more.

### MVC

Model View Controller : MVC separates web applications into 3 different components:

- Models for data
- Views for display
- Controllers for input

### WebForms

Event Driven Model :

- The traditional ASP.NET event driven development model:
- Web pages with added server controls, server events, and server code.

### Web applications and web servers

#### Web application

web application or "web app" is a software program that runs on a web server. Unlike traditional desktop applications, which are launched by your operating system, web apps must be accessed through a web browser.

Web apps have several advantages over desktop applications. Since they run inside web browsers, developers do not need to develop web apps for multiple platforms. For example, a single application that runs in Chrome will work on both Windows and OS X. Developers do not need to distribute software

updates to users when the web app is updated. By updating the application on the server, all users have access to the updated version.

From a user standpoint, a web app may provide a more consistent user interface across multiple platforms because the appearance is dependent on the browser rather than the operating system. Additionally, the data you enter into a web app is processed and saved remotely. This allows you to access the same data from multiple devices, rather than transferring files between computer systems.

While web applications offer several benefits, they do have some disadvantages compared to desktop applications. Since they do not run directly from the operating system, they have limited access to system resources, such as the CPU, memory, and the file system. Therefore, high-end programs, such as video production and other media apps generally perform better as desktop applications. Web apps are also entirely dependent on the web browser. If your browser crashes, for example, you may lose your unsaved progress. Also, browser updates may cause incompatibilities with web apps, creating unexpected issues.

## **Web Servers**

A Web server is a program that, using the client/server model and the World Wide Web's Hypertext Transfer Protocol, serves the files that form Web pages to Web users (whose computers contain HTTP clients that forward their requests). Every computer on the Internet that contains a Web site must have a Web server program.

### **Examples:**

**Microsoft's Internet Information Server, which comes with the Windows NT server;**

**Netscape FastTrack and Enterprise servers;**

**Apache, a Web server for UNIX -based operating systems.**

Web servers often come as part of a larger package of Internet- and intranet-related programs for serving e-mail, downloading requests for File Transfer Protocol files, and building and publishing Web pages. Considerations in choosing a Web server include how well it works with the operating system and other servers, its ability to handle server-side programming, and publishing, search engine, and site building tools that may come with it.

## **Application Server**

Application servers are software that help enterprises develop, deploy and manage large numbers of applications that are mostly distributed in nature.

**A Web server exclusively handles HTTP requests, whereas an application server serves business logic to application programs through any number of protocols.**

## **HTML form development**

HTML forms are used to pass data to a server.

A form can contain input elements like text fields, checkboxes, radio-buttons, submit buttons and more. A form can also contain select lists, textarea, fieldset, legend, and label elements.

The <form> tag is used to create an HTML form:

```
<form>
.
 input elements
.
</form>
```

## HTML Forms - The Input Element

The most important form element is the input element.

The input element is used to select user information.

An input element can vary in many ways, depending on the type attribute. An input element can be of type text field, checkbox, password, radio button, submit button, and more.

The most used input types are described below :

### Text Fields

<input type="text" /> defines a one-line input field that a user can enter text into:

```
<form>
First name: <input type="text" name="firstname" />

Last name: <input type="text" name="lastname" />
</form>
```

HTML code above looks in a browser as:

First name:  Last name:

**Note:** Default width of a text field is 20 characters.

### Password Field

<input type="password" /> defines a password field:

```
<form>
Password: <input type="password" name="pwd" />
</form>
```

HTML code above looks in a browser as:

Password: \$\$\$\$\$\$\$\$\$\$

**Note:** The characters in a password field are masked (shown as asterisks or circles).

## Radio Buttons

`<input type="radio" />` defines a radio button. Radio buttons let a user select ONLY ONE of a limited number of choices:

```
<form>
<input type="radio" name="sex" value="male" /> Male

<input type="radio" name="sex" value="female" /> Female
</form>
```

HTML code above looks in a browser as:

<input type="radio"/> Male <input type="radio"/> Female
---------------------------------------------------------

## Checkboxes

`<input type="checkbox" />` defines a checkbox. Checkboxes let a user select ONE or MORE options of a limited number of choices.

```
<form>
<input type="checkbox" name="vehicle" value="Bike" /> I have a bike

<input type="checkbox" name="vehicle" value="Car" /> I have a car
</form>
```

HTML code above looks in a browser as:

<input type="checkbox"/> I have a bike <input type="checkbox"/> I have a car
------------------------------------------------------------------------------

## Submit Button

`<input type="submit" />` defines a submit button.

A submit button is used to send form data to a server. The data is sent to the page specified in the form's action attribute. The file defined in the action attribute usually does something with the received input:

```
<form name="input" action="html_form_action.asp" method="get">
Username: <input type="text" name="user" />
<input type="submit" value="Submit" />
</form>
```

HTML code above looks in a browser as:

Username:

If you type some characters in the text field above, and click the "Submit" button, the browser will send your input to a page called "html\_form\_action.asp". The page will show you the received input.

## Client side scripting

JavaScript is Netscape's cross-platform, object-oriented scripting language. Core JavaScript contains a core set of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements. Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects.

**Client-side JavaScript** extends the core language by supplying objects to control a browser (Navigator or another web browser) and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.

**Server-side JavaScript** extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a relational database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.

### **Example for Client Side Scripting :**

```
<HTML> <center> <h1> Using JavaScript and HTML Forms </h1> </center>
```

```



```

```
<form action="Z:\sathyabama\SATHYA\FLDR\IP\example\html\ex1.html" method="get" name="form1"
action="javascript:void(0)">
```

```
Enter the First Value
 <input type="TEXT" name="first" size="20"
value="">

```

```
Enter the Second Value
 <input type="TEXT" name="second" size="20"
value="" >


```

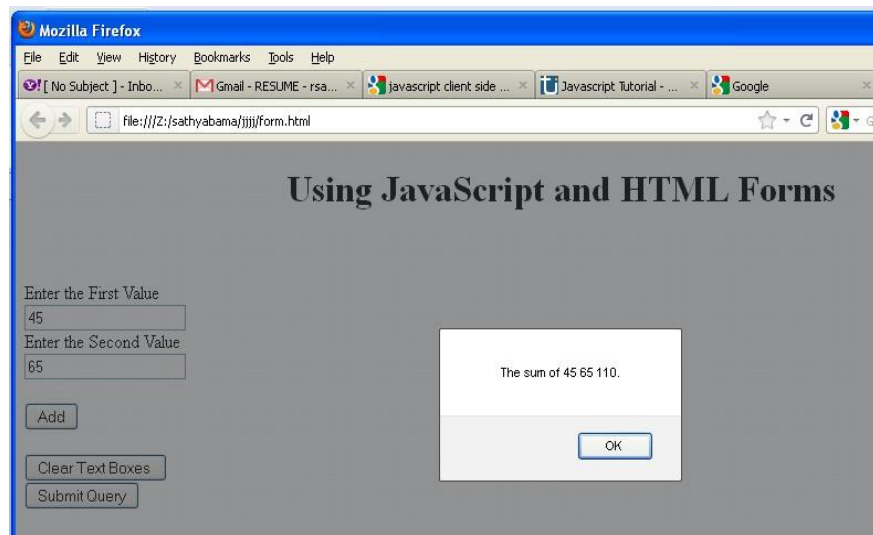
```
<input type="button" value="Add" NAME="btnbutton" onclick="add()">
```

```


 <input type="RESET" value="Clear Text Boxes " >

```

```
<input type="submit" NAME="ss"> </form>
```





## GET and POST

The method attribute specifies how to send form-data (the form-data is sent to the page specified in the action attribute).

The form-data can be sent as URL variables (with method="get") or as HTTP post (with method="post").

### Notes on the "get" method:

- This method appends the form-data to the URL in name/value pairs
- This method is useful for form submissions where a user want to bookmark the result
- There is a limit to how much data you can place in a URL (varies between browsers), therefore, you cannot be sure that all of the form-data will be correctly transferred
- Never use the "get" method to pass sensitive information! (password or other sensitive information will be visible in the browser's address bar)

### Notes on the "post" method:

- This method sends the form-data as an HTTP post transaction
- Form submissions with the "post" method cannot be bookmarked
- The "post" method is more robust and secure than "get", and "post" does not have size limitations

### Syntax

**<form method ="get | post">**

### Attribute Values

Value	Description
get	Default. Appends the form-data to the URL in name/value pairs: URL? name=value&name=value
post	Sends the form-data as an HTTP post transaction

```
<form action="form_action.asp" method="get">
 First name: <input type="text" name="fname" />

 Last name: <input type="text" name="lname" />

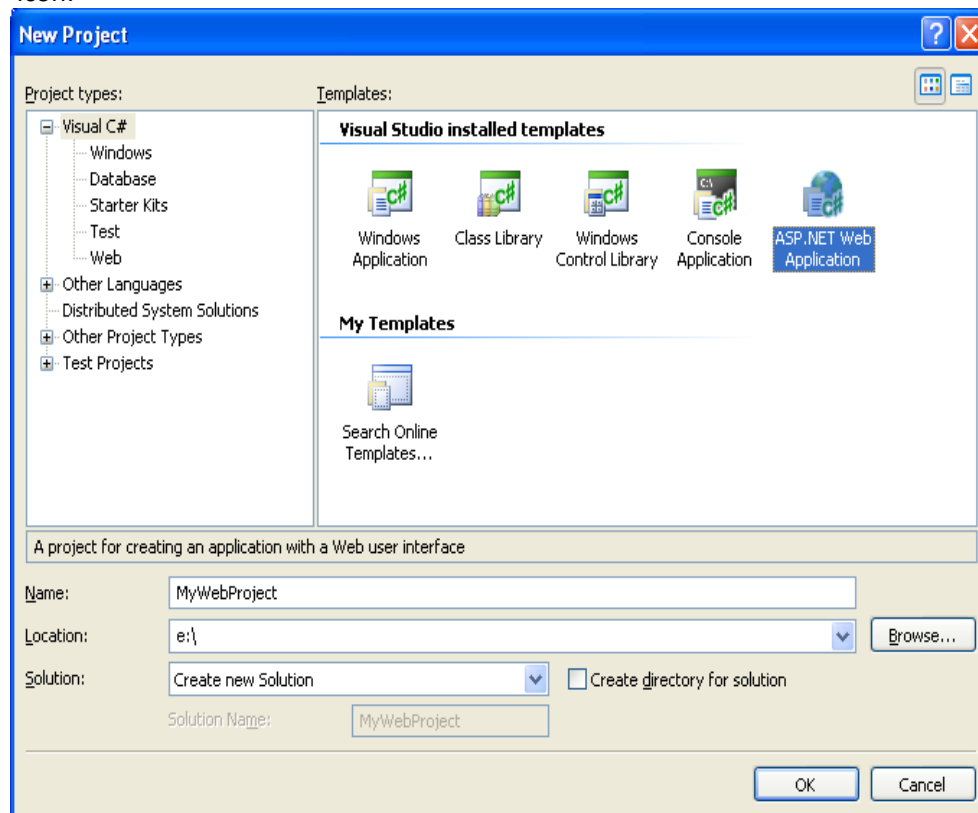
 <input type="submit" value="Submit" />
</form>
```

## ASP.NET Application

Creating, building and executing first web app using C# and the ASP.NET Web Application Project.

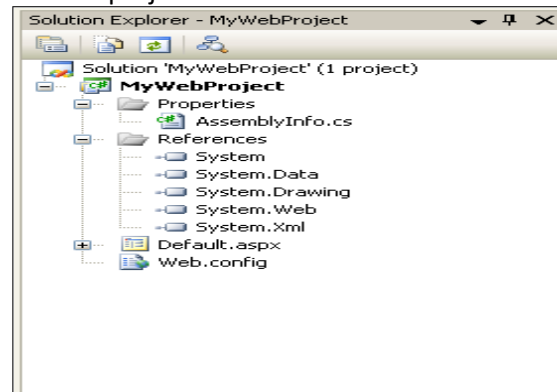
### Creating a New Project

Select File->New Project within the Visual Studio IDE. This will bring up the New Project dialog. Click on the "Visual C#" node in the tree-view on the left hand side of the dialog box and choose the "ASP.NET Web Application" icon:



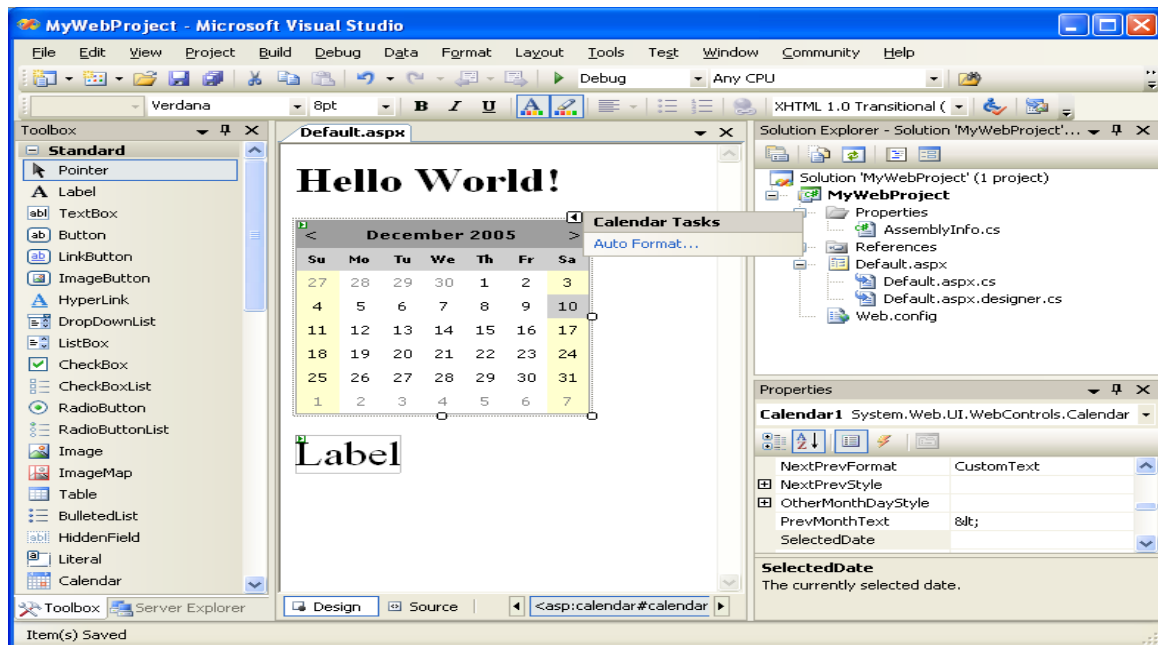
Choose where you want the project to be created on disk (note that there is no longer a requirement for web projects to be created underneath the inetpub\wwwroot directory -- so you can store the project anywhere on your filesystem). Then name it and hit ok.

Visual Studio will then create and open a new web project within the solution explorer. By default it will have a single page (Default.aspx), an AssemblyInfo.cs file, as well as a web.config file. All project file-meta-data is stored within a MSBuild based project file.



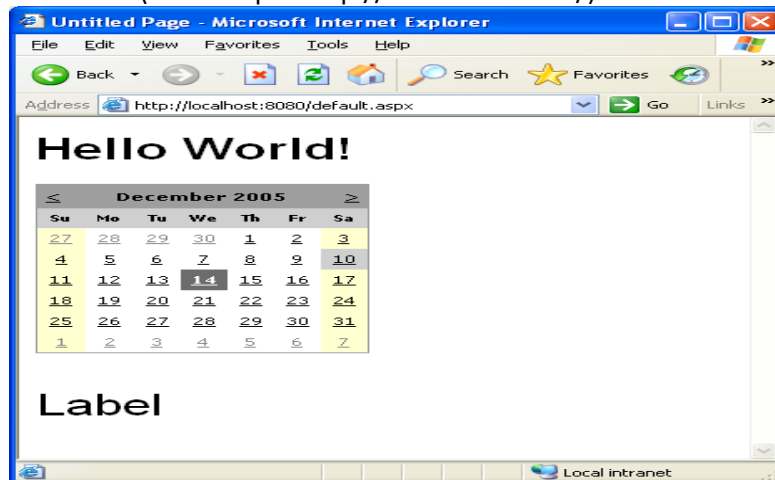
## Opening and Editing the Page

Double click on the Default.aspx page in the solution explorer to open and edit the page. You can do this using either the HTML source editor or the design-view. Add a "Hello world" header to the page, along with a calendar server control and a label control.



## Build and Run the Project

Hit F5 to build and run the project in debug mode. By default, ASP.NET Web Application projects are configured to use the built-in VS web-server (aka Cassini) when run. The default project templates will run on a random port as a root site (for example: <http://localhost:12345/>).



You can end the debug session by closing the browser window, or by choosing the Debug->Stop Debugging (Shift-F5) menu item.

## ASP.NET Namespaces

Asp .net3.5 there are well over 30 web-centric namespace in the base class libraries.

From a high level, these namespace can be grouped into several major categories.

### 1. Core functionality

2.Web form and HTMLcontrols

3.Mobile web development

4.Silverlight development

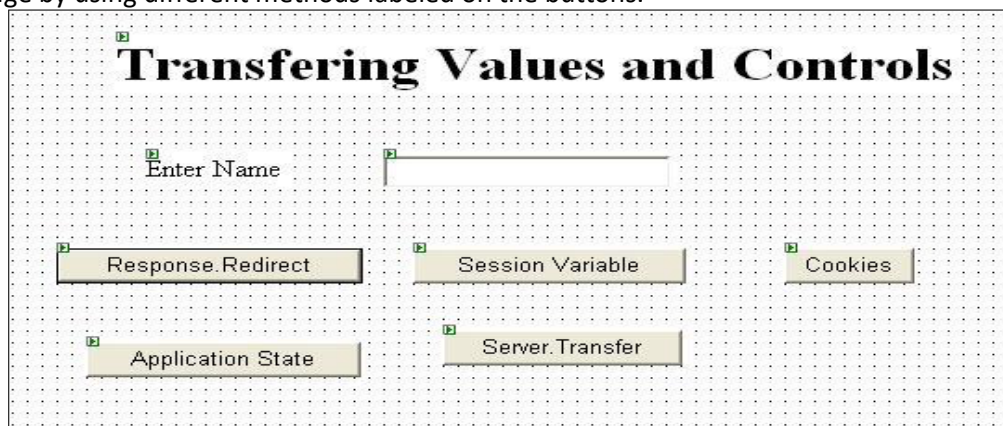
5.Ajax development

6.XML web service

NAMESPACE	MEANING
1. System.Web	defines type that enable browser/web server communication.
2. System.Web.Caching	defines type that facilitate caching support for a web.
3. System.Web.Hosting	defines type that allow to build custom host for ASP.NET runtime.
4. System.Web.Management	defines type for monitoring & managing the health of an ASP.NET web application.
5. System.Web.Profile	defines type that are used to implement ASP.NET user profile.
6. System.Web.Security	defines type that allow you to programmatically secure your site
7. System.Web.SessionState	defines type that allow you to maintain stateful information on a per-user basis.
8. System.Web.UI, System.Web.UI.WebControls, System.Web.UI.HtmlControls.	defines a number of type that allow you to build a GUI front end for your own web application.

### Creating sample C# web Applications.

We always come into situations in which we need to transfer values from one page to another page. Some different ways of transferring values from page to page. The page created in this example is really simple which consists of a text field and a few buttons. The data entered in the text field will be transferred to another page by using different methods labeled on the buttons.



### Introduction

We always come into situations in which we need to transfer values from one page to another page. In this article, I will show you some ways of transferring values from page to page. The page I created in this example is really simple which consists of a text field and a few buttons. The data entered in the text field will be transferred to another page by using different methods labeled on the buttons.

## Response.Redirect

Let's first see how to transfer using Response.Redirect method. This maybe the easiest of them all. You start by writing some data in the text field, and when you finish writing the data, you press the button labeled 'Reponse.Redirect'. One tip that I would like to share with you is, sometimes we want to transfer to another page inside the catch exception, meaning exception is caught and we want to transfer to another page. If you try to do this, it may give you a System.Threading exception. This exception is raised because you are transferring to another page leaving behind the thread running. You can solve this problem using:

```
Response.Redirect("WebForm5.aspx",false);
```

This tells the compiler to go to page "WebForm5.aspx", and "false" here means that don't end what you were doing on the current page. You should also look at the System.Threading class for threading issues. Below, you can see the C# code of the button event. "txtName" is the name of the text field whose value is being transferred to a page called "WebForm5.aspx". "Name" which is just after "?" sign is just a temporary response variable which will hold the value of the text box.

```
private void Button1_Click(object sender, System.EventArgs e)
{ // Value sent using HttpResponseMessage
 Response.Redirect("WebForm5.aspx?Name="+txtName.Text);
}
```

Okay, up till this point, you have send the values using Response. But now, where do I collect the values, so in the "WebForm5.aspx" page\_Load event, write this code. First, we check that the value entered is not null. If it's not, then we simply display the value on the page using a Label control. Note: When you use Response.Redirect method to pass the values, all the values are visible in the URL of the browser. You should never pass credit card numbers and confidential information via Response.Redirect. if (Request.QueryString["Name"]!= null)

```
Label3.Text = Request.QueryString["Name"];
```

## Cookies

Cookies are created on the server side but saved on the client side. In the button click event of 'Cookies', write this code:

```
HttpCookie cName = new HttpCookie("Name");
cName.Value = txtName.Text;
Response.Cookies.Add(cName);
Response.Redirect("WebForm5.aspx");
```

First, we create a cookie named "cName". Since one cookie instance can hold many values, we tell the compiler that this cookie will hold "Name" value. We assign to it the value of the TextBox and finally add it in the Response stream, and sent it to the other page using Response.Redirect method. Let's see here how we can get the value of the cookie which is sent by one page.

```
if (Request.Cookies["Name"] != null)
 Label3.Text = Request.Cookies["Name"].Value;
```

As you see, it's exactly the same way as we did before, but now we are using Request.Cookies instead of Request.QueryString. Some browsers don't accept cookies.

## Session Variables

Session variables which are handled by the server. Sessions are created as soon as the first response is being sent from the client to the server, and session ends when the user closes his browser window or some abnormal operation takes place. Here is how you can use session variables for transferring values. Below you can see a Session is created for the user and "Name" is the key, also known as the Session key, which is assigned the TextBox value.

```
// Session Created
Session["Name"] = txtName.Text;
Response.Redirect("WebForm5.aspx");
// The code below shows how to get the session value.
// This code must be placed in other page.
if(Session["Name"] != null)
 Label3.Text = Session["Name"].ToString();
```

## Application Variables

Sometimes, we need to access a value from anywhere in our page. For that, you can use Application variables. Here is a small code that shows how to do that. Once you created and assigned the Application variable, you can retrieve its value anywhere in your application.

```
// This sets the value of the Application Variable
Application["Name"] = txtName.Text;
Response.Redirect("WebForm5.aspx");
// This is how we retrieve the value of the Application Variable
if(Application["Name"] != null)
 Label3.Text = Application["Name"].ToString();
```

## HttpContext

You can also use HttpContext to retrieve values from pages. The values are retrieved using properties or methods. It's a good idea to use properties since they are easier to code and modify. In your first page, make a property that returns the value of the TextBox.

```
public string GetName
{
 get { return txtName.Text; }
}
```

We will use Server.Transfer to send the control to a new page. Note that Server.Transfer only transfers the control to the new page and does not redirect the browser to it, which means you will see the address of the old page in your URL. Simply add the following line of code in 'Server.Transfer' button click event:

```
Server.Transfer("WebForm5.aspx");
Now, let's go to the page where the values are being transferred, which in this case is
"webForm5.aspx".
Collapse | Copy Code
// You can declare this Globally or in any event you like
WebForm4 w;
// Gets the Page.Context which is Associated with this page
w = (WebForm4)Context.Handler;
// Assign the Label control with the property "GetName" which returns string
Label3.Text = w.GetName;
```

**Sample coding for server side script: [study your lab asp coding or the following code]**

**Default.aspx.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication3
{
 public partial class _Default : Page
 {
 protected void Page_Load(object sender, EventArgs e)
 {
 }

 protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
 {
 }

 protected void Button1_Click(object sender, EventArgs e)
 {
 if (t2.Text == "hello")
 {
 Session["NAME"] = t1.Text;
 Session["PASSWORD"] = t2.Text;
 Session["GENDER"] = r1.SelectedItem.Text;
 Session["cal"] = cd.SelectedDate;

 if (c1.Checked == true)
 {
 Session["CRICKET"] = l1.Text;
 }
 else
 {
 Session["CRICKET"] = " ";
 }
 if (c2.Checked == true)
 {
 Session["FOOTBALL"] = l2.Text;
 }
 else
 {
 Session["FOOTBALL"] = " ";
 }
 if (c3.Checked == true)
 {
 Session["CHESS"] = l3.Text;
 }
 else
 {

```

```

 Session["CHESS"] = " ";
 }

 Session["COURSE"] = cb.SelectedItem.Text;
 Server.Transfer("WebForm1.aspx");
}
else
{
 b1.Attributes.Add("onclick", "return confirm('Enter valid password')");
}
}

protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
{
}

protected void c3_CheckedChanged(object sender, EventArgs e)
{
}
}
}

```

#### **WebForm1.aspx.cs**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WebApplication3
{
 public partial class WebForm1 : System.Web.UI.Page
 {
 protected void Page_Load(object sender, EventArgs e)
 {
 t11.Text = Session["NAME"].ToString();
 t12.Text = Session["Password"].ToString();
 t13.Text = Session["GENDER"].ToString();
 t14.Text = Session["COURSE"].ToString();
 tt.Text = Session["cal"].ToString();
 tt1.Text = Session["CRICKET"].ToString();
 tt2.Text = Session["FOOTBALL"].ToString();
 tt3.Text = Session["CHESS"].ToString();
 }
 }
}

```



## Default.aspx

USERNAME

PASSWORD

GENDER

☐ MALE

☐ FEMALE

HOBBY

☐ CRICKET ☐ FOOTBALL ☐ CHESS

SELECT YOUR COURSE

CSE

March 2012

Mon	Tue	Wed	Thu	Fri	Sat	Sun
27	28	29	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

SUBMIT

## Webform1.cs

NAME: Bibin

PASSWORD: hello

GENDER: MALE

COURSE: CSE

HOBBY: CRICKET  
FOOTBALL

DATE: 15-03-2012 00:00:00

## Understanding Web Security

### ASP.NET Authentication

ASP.NET implements additional authentication schemes using authentication providers, which are separate from and apply only after the IIS authentication schemes. ASP.NET supports the following authentication providers:

- Windows (default)
- Forms
- Passport
- None

To enable an authentication provider for an ASP.NET application, use the authentication element in either machine.config or Web.config as follows:

## **Windows Authentication**

### **Configuring Windows Authentication**

To configure your application to use Integrated Windows authentication, you must use IIS Manager to configure your application's virtual directory security settings and you must configure the <authentication> element in the Web.config file.

To configure Windows authentication

- 1.Start Internet Information Services (IIS).
- 2.Right-click your application's virtual directory, and then click Properties.
- 3.Click the Directory Security tab.
4. Under Anonymous access and authentication control, click Edit.
- 5.Make sure the Anonymous access check box is not selected and that Integrated Windows authentication is the only selected check box.

In your application's Web.config file or in the machine-level Web.config file, ensure that the authentication mode is set to Windows as shown here.

```
<system.web>
 <!-- mode=[Windows|Forms|Passport|None] -->
 <authentication mode="Windows" />
</system.web>
```

Each ASP.NET authentication provider supports an OnAuthenticate event that occurs during the authentication process, which you can use to implement a custom authorization scheme. The primary purpose of this event is to attach a custom object that implements the [IPrincipal Interface](#) to the context.

Which ASP.NET authentication provider you use typically depends upon which IIS authentication scheme you choose. If you are using any of the IIS authentication schemes other than Anonymous, you will likely use the Windows authentication provider. Otherwise, you will use Forms, Passport, or None.

### **Windows**

The Windows authentication provider relies upon IIS to perform the required authentication of a client. After IIS authenticates a client, it passes a security token to ASP.NET. ASP.NET constructs and attaches an

object of the `WindowsPrincipal` Class to the application context based on the security token it receives from IIS.

- Authenticates using Windows accounts, so you do not need to write any custom authentication code.

## Con

- May require the use and management of individual Windows user accounts.

In addition, each IIS authentication scheme has its own associated pros and cons, which you should consider when choosing a security model.

## Authorizing Windows Users

When you use Windows authentication to authenticate users, you can use the following authorization options:

- File authorization provided by the **FileAuthorizationModule**.
- URL authorization provided by the **UrlAuthorizationModule**.

**Note** File authorization requires Windows authentication. The other authorization options are available with other authentication mechanisms.

```
<authorization>
 <deny users="DomainName\UserName" />
 <allow roles="DomainName\WindowsGroup" />
</authorization>
```

When you use Windows authentication, user names take the form `domainName\userName`. Windows groups are used as roles and they take the form `domainName\windowsGroupName`. Well known local groups such as Administrators and Users are referenced by using the "BUILTIN" prefix as shown here.

```
<authorization>
 <allow users="DomainName\Bob, DomainName\Mary" />
 <allow roles="BUILTIN\Administrators, DomainName\Manager" />
 <deny users="*" />
</authorization>
```

## Forms Authentication

The Forms authentication provider is an authentication scheme that makes it possible for the application to collect credentials using an HTML form directly from the client. The client submits credentials directly to your application code for authentication. If your application authenticates the client, it issues a cookie to the client that the client presents on subsequent requests. If a request for a protected resource does not contain the cookie, the application redirects the client to the logon page. When authenticating credentials, the application can store credentials in a number of ways, such as a configuration file or a SQL Server database.

**Note** An ISAPI server extension only handles those resources for which it has an application mapping. For example, the ASP.NET ISAPI server extension only has application mappings for particular resources, such as .asax, .ascx, .aspx, .asmx, and .config files to name a few. By default, the ASP.NET ISAPI server

extension, and subsequently the Forms authentication provider, does not process any requests for non-ASP.NET resources, such as .htm, .jpg or .gif files.

### **Pros**

- Makes it possible for custom authentication schemes using arbitrary criteria.
- Can be used for authentication or personalization.
- Does not require corresponding Windows accounts.

### **Cons**

- Is subject to replay attacks for the lifetime of the cookie, unless using SSL/TLS.
- Is only applicable for resources mapped to Aspnet\_isapi.dll.

### **Implementation**

To implement forms authentication you must create your own logon page and redirect URL for unauthenticated clients. You must also create your own scheme for account authentication. The following is an example of a Web.config configuration using Forms authentication:

```
<!-- Web.config file -->
<system.web>
 <authentication mode="Forms">
 <forms forms="401kApp" loginUrl="/login.aspx" />
 </authentication>
</system.web>
```

Because you are implementing your own authentication, you will typically configure IIS for Anonymous authentication.

### **Passport**

The Passport authentication provider is a centralized authentication service provided by Microsoft that offers a single logon and core profile services for member sites. Passport is a forms-based authentication service. When member sites register with Passport, the Passport service grants a site-specific key. The Passport logon server uses this key to encrypt and decrypt the query strings passed between the member site and the Passport logon server.

### **Pros**

- Supports single sign-in across multiple domains.
- Compatible with all browsers.

### **Con**

- Places an external dependency for the authentication process.

## Implementation

To implement Passport, you must register your site with the Passport service, accept the license agreement, and install the Passport SDK prior to use. You must configure your application's Web.config file as follows:

```
<!-- Web.config file -->
<system.web>
 <authentication mode="Passport" />
</system.web>
```

## None (Custom Authentication)

Specify "None" as the authentication provider when users are not authenticated at all or if you plan to develop custom authentication code. For example, you may want to develop your own authentication scheme using an ISAPI filter that authenticates users and manually creates an object of the GenericPrincipal Class.

### Pros

- Offers total control of the authentication process providing the greatest flexibility.
- Provides the highest performance if you do not implement an authentication method.

### Cons

- Custom-built authentication schemes are seldom as secure as those provided by the operating system.
- Requires extra work to custom-build an authentication scheme.

## Implementation

To implement no authentication or to develop your own custom authentication, create a custom ISAPI filter to bypass IIS authentication. Use the following Web.config configuration:

```
<!-- Web.config file -->
<system.web>
 <authentication mode="None" />
</system.web>
```

## Web services

### What are Web Services?

- Web services are application components
- Web services communicate using open protocols
- Web services are self-contained and self-describing
- Web services can be discovered using UDDI
- Web services can be used by other applications
- XML is the basis for Web services.

## **How Does it Work?**

**The basic Web services platform is XML + HTTP.**

XML provides a language which can be used between different platforms and programming languages and still express complex messages and functions.

The HTTP protocol is the most used Internet protocol.

Web services platform elements:

- SOAP (Simple Object Access Protocol)
- UDDI (Universal Description, Discovery and Integration)
- WSDL (Web Services Description Language)

## **Why Web Services?**

A few years ago Web services were not fast enough to be interesting.

### **Interoperability has Highest Priority**

When all major platforms could access the Web using Web browsers, different platforms could interact. For these platforms to work together, Web-applications were developed.

Web-applications are simply applications that run on the web. These are built around the Web browser standards and can be used by any browser on any platform.

### **Web Services take Web-applications to the Next Level**

By using Web services, your application can publish its function or message to the rest of the world.

Web services use XML to code and to decode data, and SOAP to transport it (using open protocols).

With Web services, your accounting department's Win 2k server's billing system can connect with your IT supplier's UNIX server.

### **Web Services have Two Types of Uses**

#### **Reusable application-components.**

There are things applications need very often. So why make these over and over again?

Web services can offer application-components like: currency conversion, weather reports, or even language translation as services.

#### **Connect existing software.**

Web services can help to solve the interoperability problem by giving different applications a way to link their data.

With Web services you can exchange data between different applications and different platforms.

## Web Services Platform Elements

Web Services have three basic platform elements: SOAP, WSDL and UDDI.

### What is SOAP?

SOAP (Simple Object Access Protocol) is a messaging protocol that allows programs that run on disparate operating systems (such as Windows and Linux) to communicate using Hypertext Transfer Protocol (HTTP) and its Extensible Markup Language (XML). SOAP is a protocol for accessing a Web Service.

- SOAP stands for Simple Object Access Protocol
- SOAP is a communication protocol
- SOAP is a format for sending messages
- SOAP is designed to communicate via Internet
- SOAP is platform independent
- SOAP is language independent
- SOAP is based on XML
- SOAP is simple and extensible
- SOAP allows you to get around firewalls
- SOAP is a W3C standard

### What is WSDL?

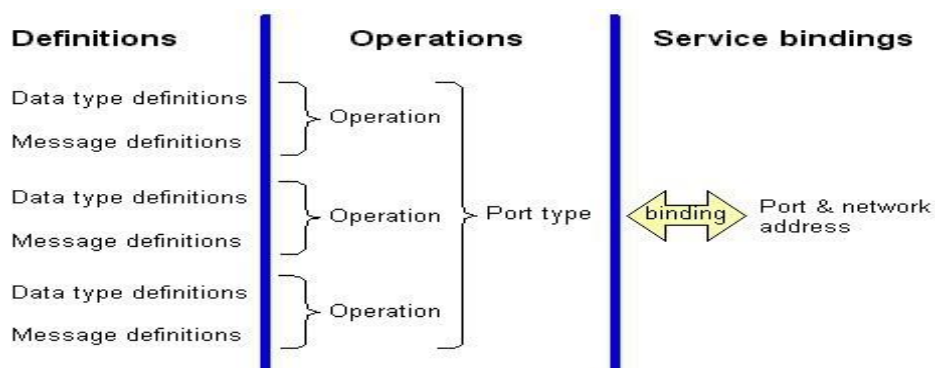
Web Services Description Language (WSDL) is a format for describing a Web Services interface. It is a way to describe services and how they should be bound to specific network addresses. WSDL has three parts:

Definitions

Operations

Service bindings

The following figure shows the relationship of the basic parts of WSDL:



WSDL is an XML-based language for locating and describing Web services.

- WSDL is based on XML
- WSDL is used to describe Web services
- WSDL is used to locate Web services
- WSDL is a W3C standard

## What is UDDI?

Universal Description, Discovery, and Integration (UDDI) provides the definition of a set of services supporting the description and discovery of (1) businesses, organizations, and other Web Services providers, (2) the Web Services they make available, and (3) the technical interfaces which may be used to access those services. The idea is to "discover" organizations and the services that organizations offer, much like using a phone book or dialing information.

UDDI is a directory service where companies can register and search for Web services.

- UDDI is a directory for storing information about web services
- UDDI is a directory of web service interfaces described by WSDL
- UDDI communicates via SOAP
- UDDI is built into the Microsoft .NET platform

## Using the Web Service Example

These functions will send an XML response like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">38</string>
```

## Put the Web Service on Your Web Site

Using a form and the HTTP POST method, you can put the web service on your site, like this:

Fahrenheit to Celsius:	<input type="text"/>
	<input type="submit" value="Submit"/>
Celsius to Fahrenheit:	<input type="text"/>
	<input type="submit" value="Submit"/>

## How To Do It

Here is the code to add the Web Service to a web page:

```
<form action='tempconvert.aspx/FahrenheitToCelsius'
method="post" target="_blank">
<table>
<tr>
<td>Fahrenheit to Celsius:</td>
<td>
<input class="frmInput" type="text" size="30" name="Fahrenheit">
</td>
</tr>
<tr>
<td></td>
<td align="right">
<input type="submit" value="Submit" class="button">
</td>
</tr>
</table>
```



```

</form>

<form action='tempconvert.asmx/CelsiusToFahrenheit'
method="post" target="_blank">
<table>
<tr>
<td>Celsius to Fahrenheit:</td>
<td>
<input class="frmInput" type="text" size="30" name="Celsius">
</td>
</tr>
<tr>
<td></td>
<td align="right">
<input type="submit" value="Submit" class="button">
</td>
</tr>
</table>
</form>

```

Substitute the "tempconvert.asmx" with the address of your web service like:  
<http://www.example.com/webservices/tempconvert.asmx>.

### **Web service clients**

One of the most simplest ways of adding a web service to a client application is by adding a web reference to the web service by specifying the URL of the *.asmx* file. This generates the required proxy object, that's what VS.NET takes care of. However, it may happen that after adding the web reference, the web service is moved to some other location. In such cases, the most easy way is to recreate the proxy object. But what if the same thing happens after you deploy your web service client. It would be nice to allow a configurable URL so that even if the original web service is moved, your client applications need not be recompiled.

Two web services are created here, one of which will have direct call and another web service will look into the web.config file for the reference.

#### **Create web service**

For our example, we will develop a simple web service that has only one method. The following steps will show you how to proceed.

- Create a new C# Web Service project in VS.NET.
- Open the default *.Asmx* file and add the following code to it:

```

using System;
using System.Web.Services;

namespace HWWebService
{

```

```

public class HWClass : System.Web.Services.WebService
{
 [WebMethod]
 public string HelloWorld()
 {
 return "Hello Application, from Web Service";
 }
}

```

As shown above, this web service class (HWClass) contains a single method called HelloWorld() that returns a string. Add another *.asmx* file to the project. Open the file and modify it as shown below:

```

using System;
using System.Web.Services;

namespace HWWebService
{
 public class AnotherService : System.Web.Services.WebService
 {
 [WebMethod] public string AnotherHelloWorld()
 {
 return "Hello World, from Another Service";
 }
 }
}

```

This class is similar to the previous one but its name is AnotherService. Also, it returns a different string from AnotherHelloWorld() method so that you can identify the method call. Now that we have both the web services ready, compile the project.

## Creating the web service client

Let us build a simple web client for our web service.

- Create a new ASP.NET web application in VS.NET.
- The application will have a default web form. Before writing any code, we need to add a web reference to our web service. Right click on the References node and select Add web reference. Follow the same procedure as you would have while developing normal web services. Adding a web reference will generate code for the proxy web service object.
- Place two text boxes. Name them as txtReturnWS and txtAnotherWS.
- Place two buttons on the web form and name them btnWS and btnAnotherWS. Add the following code in the Click event of the button:

```

private void btnWS_Click(object sender, System.EventArgs e)
{
 HelloWorld.HWServiceClass objProxy = new HelloWorld.HWServiceClass();
 objProxy.Url = GetHWSERVICEURL();
 txtReturnWS.Text = objProxy.HelloWorld();
}

```

- The above code shows how you will normally call a web service. The web reference contains information about the location of the web service.
- If you move the .asmx file after you deploy this client, it is bound to get an error. To avoid such a situation, modify the above code as shown below:

```
private void btnAnotherWS_Click(object sender, System.EventArgs e)
{
 AnotherWorld.AnotherService objProxy = new AnotherWorld.AnotherService();
 txtAnotherWS.Text = objProxy.AnotherHelloWorld();
}
```

- In above code, we have explicitly set the **Url** property of the **objProxy** class to the required .asmx file.
- You can store this URL in the **<appSettings>** section of the *web.config* file and retrieve it at run time. Now, even if you move your web service, all you need to do is change its URL in the *web.config*.
- You can add the URL of the webservice in the *web.config* file in two ways:
  1. directly by adding the required tag to the *web.config* file and
  2. by right-clicking the service used in the application, set its property to dynamic. This will automatically add the tag required for the service, you can change the name of the service to "HWServiceURL".

The following code shows this:

```
private void btnWS_Click(object sender, System.EventArgs e)
{
 localhost.HWClass objProxy=new localhost.HWClass;
 objProxy.Url=GetHWServiceURL();
 Response.Write(objProxy.HelloWorld());
}
public string GetHWServiceURL()
{
 return
 System.Configuration.ConfigurationSettings.AppSettings["HWServiceURL"];
}
```

- The *web.config* looks like this:

```
<appSettings>
 <add key="HWServiceURL"
 value="http://localhost/HWWebService/HWService.asmx" />
</appSettings>
```

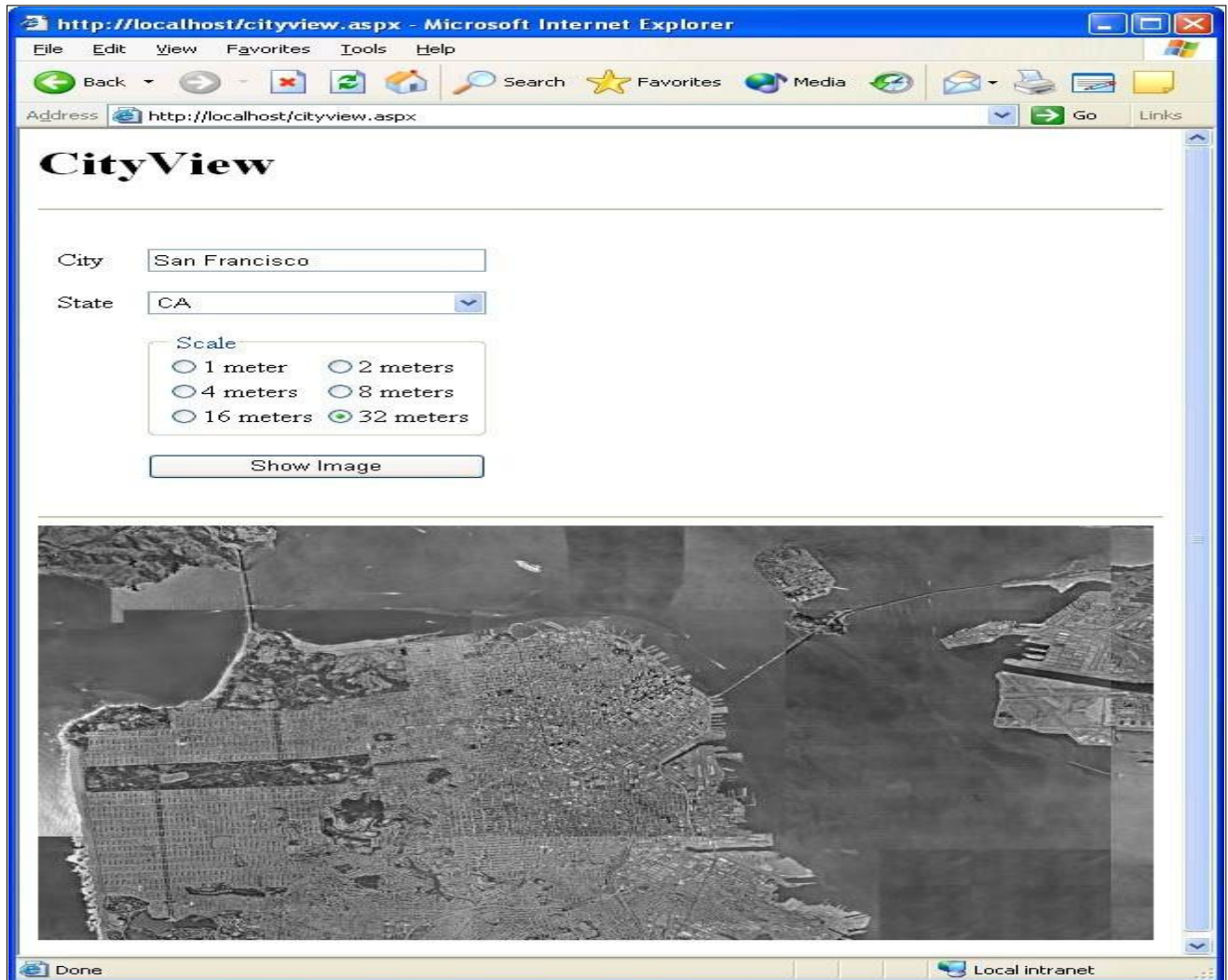
Note that in order for the above code to work correctly, both web services should have exactly same web method signatures.

### **The CityView application**

The Web application pictured in Figure CityView is a novel and graphic example of a Web service client. The Web service that it connects to is the Microsoft TerraService

(<http://terraservice.net/terraservice.asmx>), which is a Web service front end to the Microsoft TerraServer database. You can read all about TerraServer and TerraService at <http://terraservice.net/>.

TerraServer is one of the worlds largest online databases. Inside it are photographs and maps of much of Earths surface, made available to the public through a partnership between Microsoft and the U.S. Geological Survey. CityView showing an aerial view of San Francisco.



Now that CityView is installed, try it out by calling up CityView.aspx in your browser. Enter a city name (for example, New York) and pick a state. Then click Show Image. After a brief pause, the specified city appears at the bottom of the page. If CityView is unable to fetch the image you requested, it responds by displaying Image not available in place of the image. That could mean you entered a city that doesnt exist. Or it could mean that TerraService is temporarily down or your Internet connection is taking a nap.

You can zoom in and out by selecting different scales. The default scale is 8 meters. Choose a smaller number to zoom in or a larger number to zoom out. For a great aerial view of the San Francisco peninsula that includes an overhead shot of the Golden Gate Bridge, enter San Francisco, CA and choose a scale of 32 meters.

How CityView Works

CityView consists of three files:

- CityView.aspx
- CityView.ashx
- TerraService.dll

CityView.aspx is a Web form that defines CityViews user interface. Its source code appears in Figure 11-12. The user interface consists of a TextBox for typing city names, a DropDownList for selecting states, a RadioButtonList for choosing scales, and a Button for posting back to the server and fetching images. It also includes an Image control whose ImageUrl property is programmatically initialized following each postback. Heres the code that assigns a URL to the Image control:

```
MylImage.ImageUrl=?builder.ToString?();
```

If you enter Redmond, WA, and accept the default scale of 8 meters, the string assigned to ImageURL looks like this:

```
CityView.ashx?city=Redmond&state=WA&scale=8
```

which sets the stage perfectly for a discussion of the second component of CityView: namely, CityView.ashx.

CityView.ashx is an HTTP handler. Specifically, its an HTTP handler that generates and returns a bitmap image of the location named in a query string. When we deployed an HTTP handler in Chapter 8, we coded the handler in a CS file, compiled it into a DLL, and dropped the DLL into the application roots bin directory. We also registered the handler using a Web.config file. CityView.ashx demonstrates the other way to deploy HTTP handlers. You simply code an IHttpHandler-derived class into an ASHX file and include an @ WebHandler directive that identifies the class name and the language in which the class is written:

```
<%@?WebHandler?Language="C#" Class="CityViewImageGen" %>
```

When a client requests an ASHX file containing a WebHandler class, ASP.NET compiles the class for you. The beauty of deploying an HTTP handler in an ASHX file is that you dont have to register the handler in a CONFIG file or in the IIS metabase; you just copy the ASHX file to your Web server. The downside, of course, is that you must test the handler carefully to make sure ASP.NET can compile it.

The CityViewImageGen class inside CityView.ashx (Figure 11-12) generates the images that CityView.aspx displays. Its heart is the ProcessRequest method, which is called on each and every request. ProcessRequest calls a local method named GetTiledImage to generate the image. Then it returns the image in the HTTP response by calling Save on the Bitmap object encapsulating the image:

```
bitmap.Save?(context.Response.OutputStream,?format);
```

Should GetTiledImage fail, ProcessRequest returns a simple bitmap containing the words Image not available instead of an aerial photograph. It also adjusts the format of the bitmap to best fit the type of content returned: JPEG for photographs, and GIF for bitmaps containing text.

GetTiledImage uses three TerraService Web methods:

- ConvertPlaceToLonLatPt, which converts a place (city, state, country) into a latitude and longitude
- GetAreaFromPt, which takes a latitude and longitude and an image size (in pixels) and returns an AreaBoundingBox representing the image boundaries
- GetTile, which takes a tile ID (obtained from the AreaBoundingBox) and returns the corresponding tile

A tile is a 200-pixel-square image of a particular geographic location. To build larger images, a TerraService client must fetch multiple tiles and stitch them together to form a composite. That's how GetTiledImage generates the 600 x 400 images that it returns. It starts by creating a Bitmap object to represent the image. Then it uses Graphics.DrawImage to draw each tile onto the image. The logic is wholly independent of the image size, so if you'd like to modify CityView to show larger (or smaller) images, find the statement

```
Bitmap?bitmap=?GetTiledImage?(city,?state,?res,?600,?400);
```

in CityView.ashx and change the 600 and 400 to the desired width and height.

The third and final component of CityView is TerraService.dll, which contains the TerraService proxy class named TerraService. CityView.ashxs GetTiledImage method instantiates the proxy class and uses the resulting object to call TerraServices Web methods:

```
TS.TerraService?ts=?new?TS.TerraService?();
```

TerraService.dll was compiled from TerraService.cs, which I generated with the following command:  
wsdl?/namespace:TS?http://terraservice.net/terraservice.asmx

The namespace was necessary to prevent certain data types defined in TerraServices WSDL contract from conflicting with data types defined in the .NET Framework Class Library. Once TerraService.cs was created, the command

```
csc?/t:library?terraservice.cs
```

compiled it into a DLL.

TerraService exposes TerraServers content via Web methods. There are 16 Web methods in all, with names such as ConvertPlaceToLonLatPt and GetTile. As you might expect, TerraService was written with the Microsoft .NET Framework. Its WSDL contract is available at <http://terraservice.net/terraservice.asmx?wsdl>.