.Net Framework

- CLR (Common language Runtime) → MS IL (intermediate lang.) → JIT
- ASP.net
-

Namespace: imports (VB.net) / Using (C#) / Using Namespace (VC++.net) } to reference namespace

Assemflies: - physical file / unit of organization
- use complise metadata (menifest file)
- self describing
- no registry needed
- manifest file.

System.XML

* integrating with VSS (Visual Source Safe)

- save projects in DB
- control versions of project files
- add metadata to the files in the file system
- show history of project file operations
- build reports of project history
- search

✓ File / Source control / Add solution to source control / Microsoft Visual SourceSafe

Check out..
check in..

** ADO.net

System.Data

Connection Object ← provider / connection string

```
dim objConn As SqlClient.SqlConnection
objConn = New SqlClient.SqlConnection()
objConn.ConnectionString = "Integrated Security=True;" &. _
"Data Source=local Host;Initial Catalog=Pubs;"
objConn.Open()
```

* Command Object : – execute commands like stored procedures
– return data
– use SqlCommand with SQL Server database
– use OleDbCommand with other data sources

```
dim objConn As SqlClient.SqlConnection
- - - -
- - -

Dim objCommand As SqlClient.SqlCommand
objCommand = New SqlClient.SqlCommand()
objCommand.Connection = objConn
objCommand.commandText = "SELECT * FROM title"
```

* dataReader object : – retrieve read-only, forward-only streams of data
– to instantiate the DataReader
* use the ExecuteReader method of the command object
* call the Read method to obtain data in the rows

* DataAdapter Object : – pass data between a data source and a dataSet
– retrieve data for a DataTable or send updates back to the data source
– create a DataAdapter by : – using a connection object
– specifying connection information

```
Dim objConn As SqlClient.SqlConnection
- - - -
- - -

objCommand.commandText = "SELECT * FROM titles"
Dim objAdapt As SqlClient.SqlDataAdapter
objAdapt = New SqlClient.SqlDataAdapter()
objAdapt.SelectCommand = objCommand
```

* Filling A DataTable =

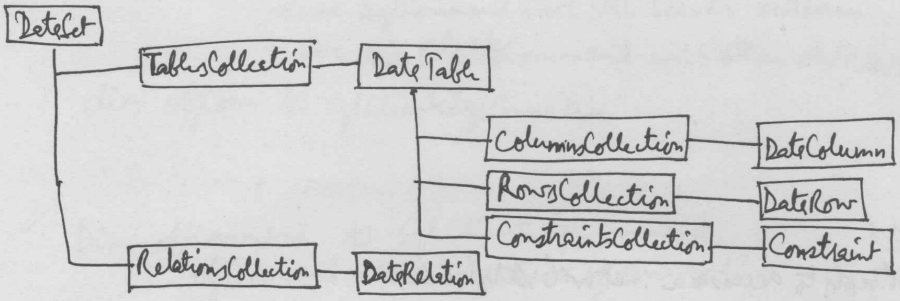
```
dim objConn As SqlClient.SqlConnection
- - - -
objConn.Open()
dim objAdapt as new SqlClient.SqlDataAdapter("SELECT * FROM Titles",
                                             objConn)

dim objDataSet As DataSet = New DataSet()
objAdapt.Fill(objDataSet, "MyTable")
objAdapt.Update(objDataSet, "MyTable")
```

eb app & = Web Services =

Consuming web services:

- use one of the three protocols to access a .net web service:
  1. HTTP - Get
  2. HTTP - Post
  3. SOAP (Simple Object Access Protocol)

DataSet Object Model:



= Object-orientation =

Constructor: public sub new() →
initialize: InitializeComponent()
destructor: public sub Dispose()
Form_Closed(), event

.Net programming in C# {- OOP language
- combines computing power of C++ with the ease of VB and Java.

.Net framework :- [code library sits underneath all .Net enabled apps.]
(makes RAD Rapid App. Dev. of highly portable code)

CLR – Common lang. Runtime
CTS – Common Type System
CLS – Common Language Specification
} CLR environment

Assemblies := (binary files) = (.DLL / .Exe)
- a new type of file format
- composed of CIL (MSIL) and metadata (manifest)
common intermediate lang.

① single file assemblies
② multifile "

Types = (store references to data)

Reference type ← System.Object → Value type

Reference type (referencing memory location of data):
- class
- Interface
- Array
- string
- Delegate → like function pointer
- others

Value type:
- signed: (S)Byte, Char, (U)Int16, (U)Int32, (U)Int64
- Boolean, Single, Double, Enum, Struct, Decimal, Others
- Float

```
using System;
namespace ...
{
  class ...
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello world!");
    }
  }
}
```

```
for (int i; i < args.Length; i++)
    Console.WriteLine("Arg: {0}", args[i]);
Console.WriteLine("Hello world");
```

// /** /* */ | (Object Browser) ✓

Literal : double Rate = 0.06M
uint i = 22U;
bool b = true;

---

explicit type conversion : x = (int) y

implicit " " : (narrowing / widening)

Value types : - their values allocated on the stack in memory. (e.g.; int i,x;
long l;
struct s{ ... }
enum e{ ... }
- can not be null (must contain data)

Reference types : - allocated on the managed heap
- can consume significant resources
- more features & benefits
- can be null
- only a reference to the object is passed when passed in methods (by reference).
so, changes to the referenced data changes the source.

(e.g.; class c{ }
interface Intf{ }
string s;
array = long[] a;
delegate void Process();)

address → reference

---

boxing = value type into → reference type e.g.; int i=20;
object objinteger = i;

unboxing = reference type into → value type int ie = (int) objinteger;

Assembly ⟺ Namespace
(physical) ↔ (logical)

Private assemblies : deployed with an application and is available for exclusive use in that application (other applications don't share the private assemblies)
Global " (strong named) : identified by globally a unique name and version (digital signature to ensure that it cannot be tempered with)
Shared " : available for use by multiple applications (shared side by side assemblies are not registered globally)

public readonly string x;

---

public short x
{
get { }
set { }
}

property

- private = access by the class only
  protected = " " " " and any derived classes
  internal = internal to the assembly only
  protected internal = access by assembly or derived classes from a class in the assembly in other assembly

static = we don't need to instantiate ...

---

enum RGBValues : byte { Red = 3, Green = 1, Blue = 2 }

```
foreach (string s in args)
    - - - ;

switch (i)
{
    Con - - :
        break;
    - - -
    default:
        - - -
        - - ;
}
```

ADO.Net;

① new Connection:

```
SqlConnection conn = new SqlConnection("Server= localhost; ---
    - - - ");
```

② Open connection:

```
conn.open()
```

③ new Command:

```
SqlCommand cmd = new SqlCommand();
String s = "SELECT* - - - - ";
Cmd.Connection = conn;
Cmd.CommandText = s;
    - - -
```

System.String — implements the functionality of string data type
— Immutable: once an instance is created, it cannot be modified.

System.Text.StringBuilder

— StringBuilder sb = new StringBuilder ("Hello World!");
— Mutable: you can modify strings in memory (e.g. insert, append, replace, remove, ...)
— starts with 0 as index

delegate : example of a reference type

= allows to reference a method (like function pointers)

— a flexible, reusable strategy to implement different functionality

representative

ex: → public delegate void sendMeAStringMethod(string s);

ex:

```
class ...
{
    public static bool MethodToCall(string s) { ... }
    public delegate bool Delegate2(string s);
    static void Main(string[] args)
    {
        Delegate2 del;
        del = new Delegate2(MethodToCall);
        del("Hello");
    }
}
```

same signature & type

— events and delegates work together for functionality

Event:

ex: public event MyDelegate MyEvent;

MyEvent += new MyDelegate(OnMyEvent); ≡ Button.Click += new EventHandler(onButtonClicked);

MyEvent() // fire!

≡ Application Class methods =

AddMessageFilter(), RemoveMessageFilter(), DoEvent(), Exit(), ExitThread(), OLERequired(), OnThreadException(), Run(), ApplicationExit(), Idle(), ThreadExecution(), ThreadExit(), ...

= **interface** : — contains definitions of: events, indexers, method, properties

— inherited by classes, structs and other interfaces

— define a contract between objects derived from these interfaces

= you implement an interface and use it in a class

ex:

```
interface IPace
{
    void Method1();
}
```

```
class Cl : IPace, ...
{
    static void Main()
    {
        Cl c = new Cl();
        c.Method1();
    }
    public void Method1()
    {
        ...
    }
}
```

}