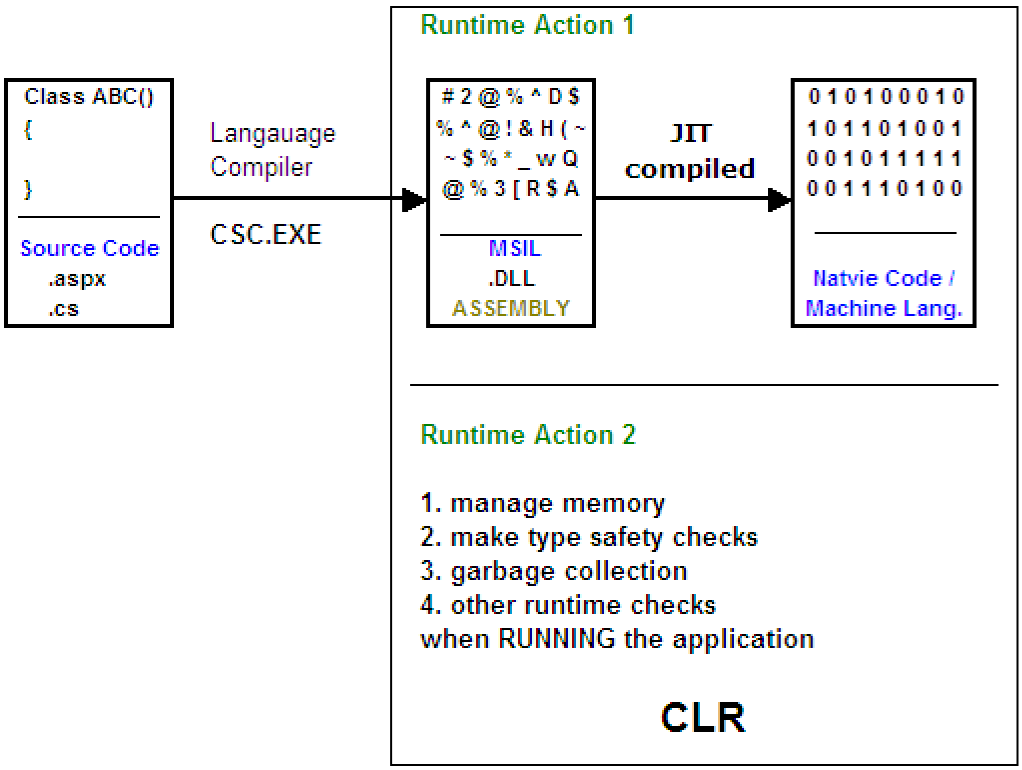
Introduction

.NET Framework has 2 main components

- **CLR** (Common Language Runtime) - provides managed runtime environment [Execution Engine]
- **FCL** (Framework Class Library) - is a collection of reusable types (classes) organized in hierarchical namespaces. It provides a library of standard types.

LANGUAGE COMPILERS compile the SOURCE CODE to IL (Intermediate Language) and store it in a file called an ASSEMBLY [IL code + metadata] → CLR, JIT-compiles the IL to MACHINE SPECIFIC NATIVE CODE at runtime.

BROWSER ←→ IIS (inetinfo.exe) ←→ aspnet_isapi.dll ←→ aspnet_wp.exe



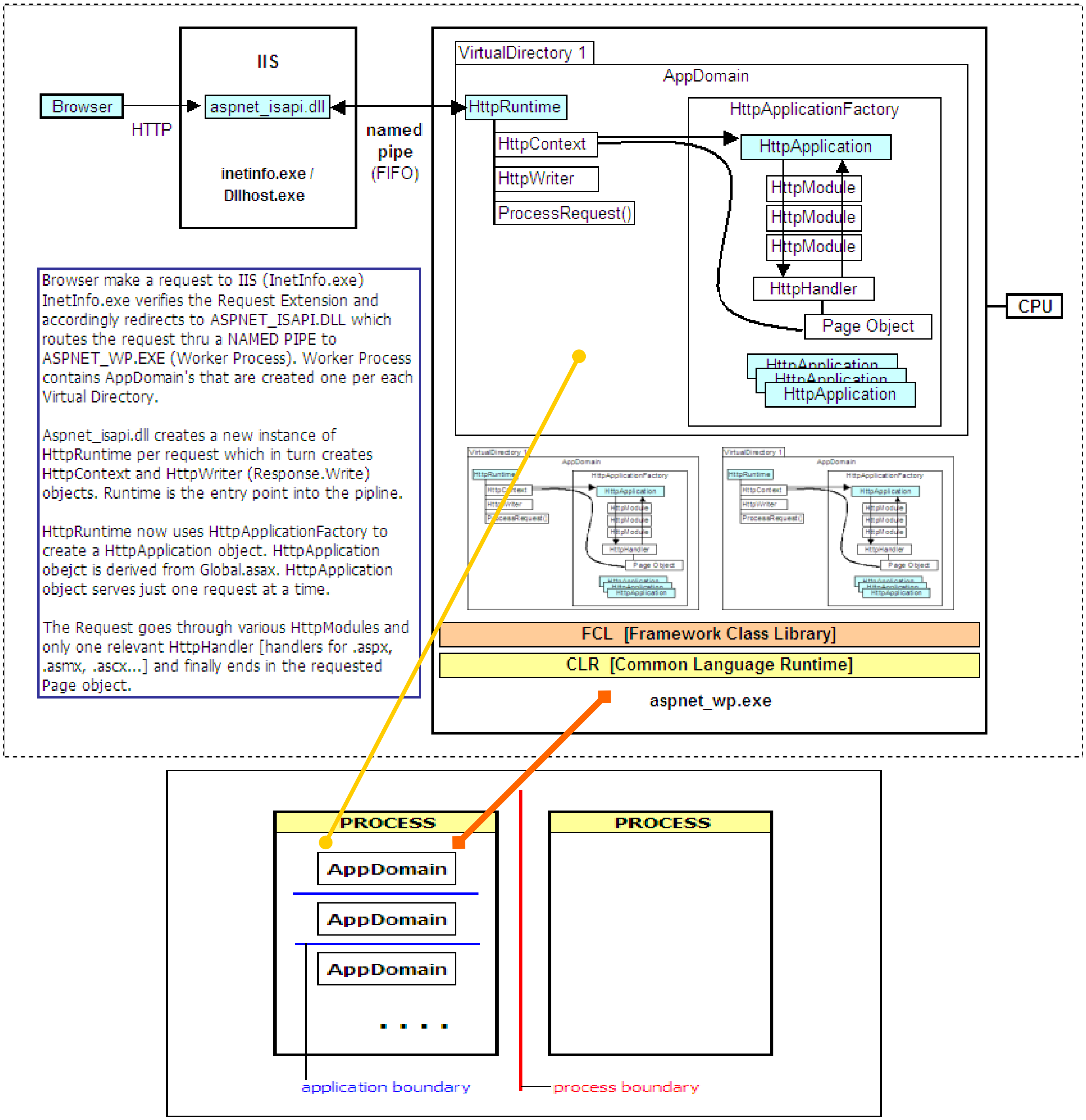
ASP.NET runtime host (Worker Process **aspnet_wp.exe**) uses FCL, loads CLR into the process, compiles the .aspx page into an assembly, creates an application domain (AppDomain) and then loads & executes the requested page within the application domain.

CLR – is an execution engine that handles runtime services such as language integration, code compilation, security, memory allocation, thread management, and garbage collection. CLR also loads, executes the IL code for running an application and while running, the CLR provides memory management, type-safety checks and other run-time tasks for the application. Loads, compiles, executes IL Code and enforce security, type safety and thread support.

FCL – provides reusable types that are designed to integrate with CLR.

- Assemblies are the smallest unit to which .NET grants permissions.
- Assemblies can be STATIC or DYNAMIC.

- Static assembly is created when you compile the program using .NET language compiler. It is stored as an .exe or .dll
- Dynamic assemblies are created at runtime when the application requires them.
- Private assemblies / Shared assemblies (GAC – Global Assembly Catalog) | sn.exe (Strong Name Tool)



DataTypes

1	bool	byte	
2	char	short	
4		int	float
8		long	double object
16			decimal
20+	string		

1	2	4	8	16	20+
byte bool					
	short char				
		int float			
			long double		
				decimal	
					string

.NET pages extensions

.aspx	asp.net page
.ascx	user control
.asmx	web service
.ashx	httphandlers
.axd	trace.axd
.asax	global.asax
.config	web.config

- A single copy of worker process runs all the time and hosts all the active web applications.
- Web Garden – multiple CPU
  Web Farm – many computers
- An AppDomain is created within the worker process whenever a client addresses a virtual directory for the first time.
- An application pool is a blanket term that identifies a worker process and a virtual directory.
- Asp.net runtime activates the HTTP pipeline by creating a new instance of the HttpRuntime class and then calling the method ProcessRequest. The HttpRuntime object starts working to serve a page to the browser. It creates a new context for the request and initializes a specialized text writer object, which is an instance of the HttpWriter class, and this object actually buffers text sent out through the Response object. A context is given by an instance of the HttpContext class. The HttpRuntime object is then used to find and create a new web application in HttpApplicationFactory. The HttpApplicationFactory object maintains a pool of HttpApplication objectsto serve the incoming HTTP requests. When invoked, the application factory object verifies that an AppDomain exists for the virtual folder the requests targets. If the application is already running, the factory picks up and HttpApplication out of the pool of available objects and passes the request. A new HttpApplication object is created if an existing object is not available. A HttpApplication object is used to process a single request at a time.
- HttpApplicationFactory maintains a pool of HttpApplication objects.
- An HttpApplication object is used to process a single request at a time. It is a global.asax derived object.
- Each asp.net is allowed a maximum number of recompiles (15 as default) before the whole application is restarted.
- When a page is changed, the assembly is recompiled and asp.net tries to delete the old assembly if asp.net finds that the old assembly is still locked then the old assembly is renamed with a .DELETE postfix. This .DELETE old assembly is scavenged during the next SWEEP MODE or whenever the application is restarted or an application file (global.asax, web.config) changes.
- Page life cycle OnInit() → viewstate property is restored → loads postback data → OnLoad() → OnPreRender() → save viewstate → OnUnLoad() [Dispose]

- Server form – default value for method=POST, HTML form – default value for method=GET
- Viewstate is the collection of the values in a Base64 string that need to be preserved during page postbacks.
- Viewstate string is deserialized and transformed into an instance of StateBag object.
- For service side events to hidden form variables are created _EVENTTARGET à contains the name of the control that caused the postback and _EVENTARGUMENT à contains any argument that might be useful to carry out the call.
- Page intrinsic objects à Application, Request, Response, Session, Server, Cache, Trace
- A master page is a distinct file referenced at the application level as well as page level that contain the static layout of the page.

Stages of ASP.NET page processing

Init — creates and instance of ASP.NET page
starts instantiation and initializing of Page and Controls
appropriate place for attaching a delegate for event handling

Load — at this stage Page Initialization is complete and the Page is loaded into memory.
user code initialization / controls are loaded in the Page object.
ViewState is available at this point.
always use this method for loading the controls dynamically.
best place for initializing controls

PreRender — pre-render / this is the last chance to modify the Pages output

Unload — page clean up. Page has been rendered and is ready for discarding.

Access modifiers
----
public - globally accessible
private - access only to the containing type
protected - access to containing type + derived types
internal - access within current project

- Asp.net applications are compiled pieces of code.
- Reentrant Form is am html <form> that posts to the same page that contains it.
- Call context are hidden fields that carry session state.
- <FORM> tag is the only element authorized to transmit client side data to the server.
- Server <form> default method="POST"
- Server <form> default method="GET"
- Objects like REQUEST, RESPONSE, SERVER form the HTTP CONTEXT for the call.
- Directives let you import namespaces, load assemblies, register new controls, etc.,
- <@ Page> sets up the environment in which the page will run. Page directive can be used only with an .aspx page.<@ Page|Control="" CodeBehind="test.cs" Src="test.cs"> Page and Control attributes are mutually exclusive. AspCompat="true|false" for STA [Single-Threaded Apartment] execution.
- Inherits attribute give the link info at runtime,
- CodeBehind attributes give the link info during design time.
- Inherits + Src attribute combination is used for on demand / dynamic compilation
- runat attribute promotes the server-side tag to the rank of component instance.

- Custom controls are created as assemblies [.dll]. User controls are source files [.ascx]
- When asp.net page is parsed all directives attributes are extracted and stored in a Dictionary.
- <@ Assembly > links an assembly to the current page. <@ Assembly Name|Src="">
- By default 9 assemblies are linked and also all the assemblies that lie in the BIN directory, this is made possible through <add assembly="*" > of machine.config.
- <@ Import > links the specified namespace to the page to allow use shorter class names instead of using fully qualified name.
- <@ Reference > is used to establish a dynamic link between the current page and the specified page. <@ Reference Page|Control="" >
- <% inline code %> <%= inline expression %> The inline expression is basically a shortcut for Response.Write().
- Only controls that are a part of the <form> can persist their state across page requests.
- When page is loaded, asp.net runtime parses the source code and creates instances of all controls marked with the runat attribute.
- Class is a Reference Type that defines a blue print of an idea (template). It contains Data (Constants and Fields) and defines its behavior through Methods, Properties, Constructs and Events.
- Objects are created from classes and have physical memory allocated to them. Objects are created on a Heap.
- Object pointers and are stored on Stack.
- Structs are Value Types similar to classes but uses Stack memory. Value Types cannot be inherited.
- Property provides access to the characteristics of the class / object. Property is not a storage location.
- Namespaces allow you to organize classes into logical groups. This will also help in avoiding naming conflicts. System namespace is the root namespace for all base classes defined in FCL.
- Response.Output.Write() allows formatted output compared to Response.Write()
- ASP.NET supports only one language for a page.
- A solution groups one or more projects.
- A directive can be placed anywhere in a page but is usually placed at the top.
- Application variables must be initialized in C#.
- src - will allow dynamic compilation of the class at runtime
- A protocol is a set of rules that describe how two or more items communicate over a medium, such as the Internet.
- Applications that run under the CLR are called managed code because the CLR takes care of many of the tasks that would have formerly been handled in the application's executable itself. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL hell) because the assembly contains all the versioning and type information that the CLR needs to run the application. The CLR handles registration dynamically at run time, rather than statically through the system registry as is done with applications based on the Component Object Model (COM).
- State variables are ApplicationState, SessionState and ViewState variables.
- Application variables must be initialized in C#.
- To read one web form's ViewState from another, you must firt set the EnableViewStateMac attribute in the web form's Page directive to false.
  <%@Page ...... EnableViewStateMac="false" ....%>
- Transfer and .Execute works with web forms only, trying to navigate to an HTML page using one of these methods results in a run-time error.
- Visual studio does not support CodeBehind if it is implemented through the Src attribute.
- ISAPI extensions (.aspx, .cs, .asmx ...) are run-time modules that handle web resources.

- .aspx files are assigned to ASPNET_ISAPI.DLL module of IIS.
- IIS5.0 INETINFO.EXE hosts Aspnet_isapi.dll.
- Aspnet_isapi.dll does not process the .aspx files but acts as a dispatcher and it routes the request toward asp.net worker process (aspnet_wp.exe)
- Resource mappings are stored in the IIS metabase.
- Configuration can be carried out like: multiple CPU's can handle 1 WORKER PROCESS / 1 CPU runs 1 aspnet_wp.exe (worker process).
- Process Recycling: <processModel> element of the machine.config file defined threshold values for all these parameters. The aspnet_isapi.dll checks the overall state of the current worker process before forwarding any request to it. If the process breaks one of these measures of good performance, a new worker process is started to serve the next request. The old process continues running as long as there are requests pending in its own queue. After that, when it ceases to be invoked, it goes into idle node and is then shut down. In this way, memory leaks and run-time anomalies are promptly detected and overcome.
- Assemblies generated for the asp.net page are cached in the temporary asp.net files folder WINDOWS\Microsoft.NET\Framework\v1.1.4322\Temporary ASP.NET Files
- an Assembly contains the IL code and metadata.
- Assemblies are the deployment units in the .NET framework.
- The code that runs within the CLR is called managed code.
- garbage collector divides the objects on the managed heap into 3 generations 0, 1, 2. Generation 0 contains recently created objects. GC first collects the unreachable objects in generation 0. Next, the gc compacts memory and promotes the reachable objects to generation 1. The objects that survive the collection process are promoted to a higher generation.
- gc cannot clean the system resources used by managed objects eg. File handles. Therefore you need to explicitly release these resources by providing the cleanup code in the Dispose().
- Dispose() for MANAGED CODE – System Resources
- gc provides finalization() for cleaning up unmanaged objects. Finalization() allows an object to perform cleanup tasks automatically before gc starts. The finalize method ensures that even if the client does not call the Dispose() explicitly, the resources used by the object are released from memory when the gc starts. Once gc identifies and object as garbage, gc during gc process calls the finalize() before releasing memory.
- Dispose() and finalize() are called finalizers.
- GC – Garbage Collection – when gc starts running, it navigates the application root list and checks for directly/indirectly referenced objects and flags these objects as reachable. The rest of the objects on the managed heap are removed and the reachable objects are copied to a new contiguous location, after this the GC updates the pointers in application root list.
- Dispose() of the object should be used to release system resources such as file handles, network connections, etc.,
- Finalize() ensures that even if the client does not call the Dispose() explicitly , the resources used by the object are released from memory on GC
- namespace groups logically and functionally related classes.
- a portable executable file cannot be executed if it does not contain an assembly manifest. The runtime uses the information in the assembly manifest to load the assemblies into the runtime.
- an assembly that is shared by multiple applications is called global assembly.
- static assembly is stored as a portable executable (.exe or .dll) file.
- dynamic assemblies are created at runtime when the application requires them. Reflection API allow you to find type info and create objects dynamically at runtime.
- private assembly is installed in the installation directory of an application.

- shared assembly is called GAC – global access cache.
    - a strong name consists of an assembly name, version, culture info, digital signature and public info.
- CLR can terminate an AppDomain without stopping the window process
- It is much cheaper to create, monitor and maintain an AppDomain than a Process.
- An application boundary / AppDomain defines the scope of an application
- A windows process runs only one application.
- Windows achieves isolation thru process boundaries.
- AppDomain is the basic unit of isolation for running application in CLR and an AppDomain achieves isolation through application domain boundaries
- CLR allows several AppDomains to run within a single window process
- AppDomain contains its own set of code, data and configuration settings

.NET framework tools

- Assembly Linker – is used for modules, AL Tool generates a file with an assembly manifest from the modules or resources file.   C:\> al /t.exe /out:abc.exe abc.dll
- IL Assembly/Disassembler – ILASM.exe / ILDASM.exe
  ILASM.exe is used to generate a portable executable file from MSIL code
  ILDASM.exe is used to view the metadata and dissembled code of PE file
- Code Access Security Policy Tool – CASPOL Tool – allows you to grant and modify permissions granted to code groups at user-policy, machine-policy and enterprise-policy level.
- Configuration Tool – MSCORCFG.msc – is a Microsoft management console MMC snap-in that enables you to manage and configure assemblies located in GAC.

Controls

HTML controls

- simple html tags,
- they do not maintain their state across postback

HTML Server controls

- html tags with runat="server"
- can be accessed on the web server
- provided for migration old code to .NET
- ServerClick / ServerChange are the default events

Web Server controls

- provide higher level of abstraction
- preferred controls
- have built-in automatic browser detection capabilities
- support event bubbling
- for asp:button if the CommandName is provided then the Button is Command Button. Command button is useful when you want to pass some info to the EventHandler.
- EventHandling
  - Intrinsic events – Click() / Command() +
  - Event Arguments
  - AutoPostBack
  - Bubbled Events
- Client side event handling
  - btnSubmit.Attributes.Add("onMouseOver", "SomeClientJSCode();")
  - Validation controls
- client side validation
  - WebUIValidation.js
  - RegisterClientScriptBlock() / IsClientScriptBlockRegistered()
  - Validation check fires onFocusOut() of the control
- server side validation
  - o Page.IsValid() / Page.Validate()
- Types of validators
  - RequiredFieldValidator - value cannot be empty
  - RegularExpressionValidaotr - validate against a RE
  - RangeValidator - data should be within a range
  - CompareValidator - compare data against a control value
  - CustomValidator - custom logic to validate data
  - ValidationSummary

User controls
Composite controls
Custom controls

Server control events

Postback events - These events cause the Web page to be sent back to the server for immediate processing.
[causes the page to send the request to the server, but their event procedure is processed last in order of events handled]

Cached events - These events are saved in the page's view state to be processed when a postback event occurs.
[these events are collected while the page is displayed and then processed once the page sends a request to the server]

Validation events - These events are handled on the page without posting back or caching.

[occur before the page is returned to the server]

There can be only one <form runat="server"> control on a web page.

Intrinsic objects

Page.IsPostBack – checks if the page is requested the first time or not.

SmartNavigation
- Persists element focus and scroll position between postbacks.
- Eliminates page flash by using double buffering technique.
- Prevents each postback from being saved in the browser history.
- In web.config <pages SmartNavigation=true />

Intrinsic objects allow ASP.NET to enable low-level access to the web application framework.

Types of intrinsic object	property of page class
HttpRequest	Request
HttpResponse	Response
HttpServerUtility	Server
HttpApplicationState	Application
HttpSessionState	Session

HttpContext.Handler / HttpContext.Items can be used to access one pages variables from another page. Context object gives access to the current HTTP request and response. CONTEXT object exposes a key-value collection via the ITEMS property in which you can add values that will be available for the life of the current request (like REQUEST scope of coldfusion).

State Management is the process of maintaining state for a web page across roundtrips. Various techniques used for maintaining client-side state management.
- client side state management
    - Query String
    - Cookies
        - Persistent cookies
        - Non-persistent cookies
    - Hidden Fields
        - HttpInputHidden control is not available as a web server control
        - Because asp.net uses a similar but more powerful technique - ViewState
    - View State
        - Possible through hidden variables
        - _viewstate hidden variable
        - Base64 encoding
        - Decodes the string at init()
        - MAC (Machine Authentication Check)
- server side state management
    - * session

Response.Redirect()
- is client side redirection.
- Can connect to any .aspx pages / non-.aspx resources on any server
- Can pass info through URL
- creates a 302 response header (object moved) status code + target url

Server.Transfer()
- is server side redirection.
- Can be used for .aspx pages only
- .aspx page must be on the same server and should not contain URL
- to preserve the original page data - Server.Transfer("a2.aspx",True) but in this situation keep the EnableViewStateMac=false on the destination page

Server.Execute()
- executes an .aspx within the current requested page.
- Similar to a method call
    Need to keep the EnableViewStateMac=false on the called page

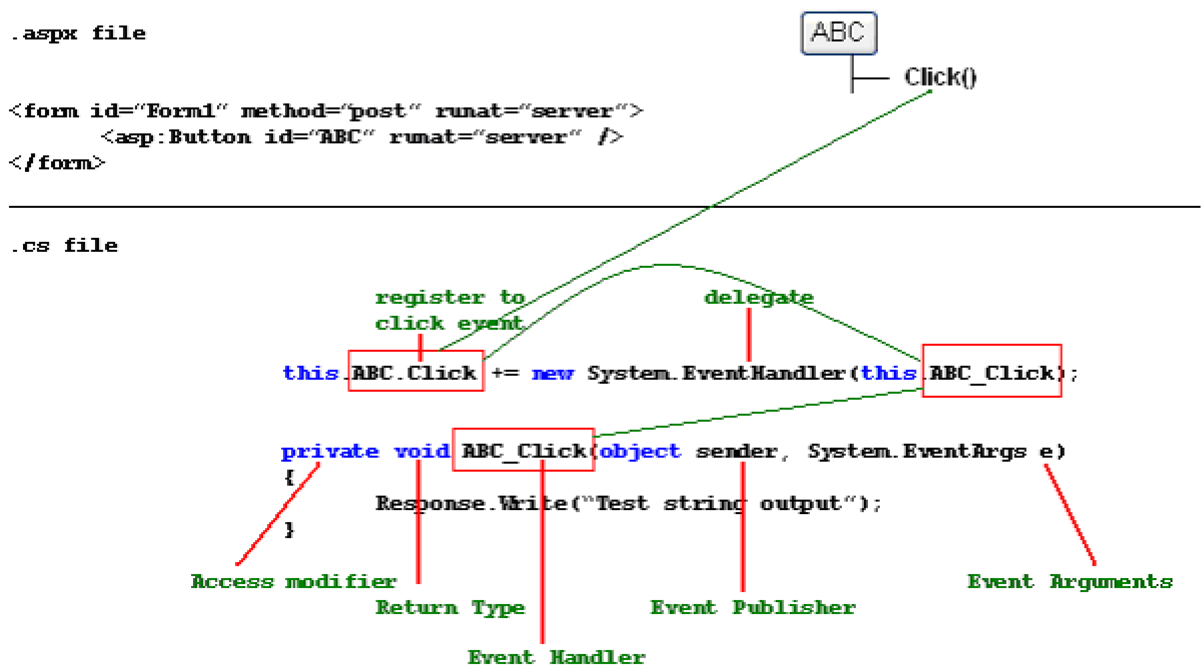
Events

- Intrinsic events
- Event arguments
- AutoPostBack
- Bubbled events
- Server control events
    - Postback events
    - Cached events
    - Validation events

event handling

- overiding virtual, protected methods of base class
- using delegates
- using AutoEventWireup

AutoEventWireup event-handling is not as flexible as event handling through delegates because the event handlers are required to have specific names. Therefore the AutoEventWireup is set to false in VS.NET.



Delegates

- Delegate is a special class [reference type] whose object is capable of storing references to methods with a particular prototype, i.e., it defines the signature for a method call.
- Delegates allow you to attach a single event handler to several events.
- OnInit() is the appropriate place for attaching a delegate for event handling.

```
// define the delegate
public delegate void DelXYZ (Object sender, EventArgs e);

// define the event
public event DelXYZ TestEvent;

// register the event with the delegate
this.TestEvent += new DelXYZ (ABC);

// notify the registered object about the event
public virtual void OnTestEvent(EventArgs e)
{      // if any object is registered with the event
       if (TestEvent != null)
       {      // raise the event | notify that object
              TestEvent (this, e);
       }
}
// some EventHandler
public void ABC (Object sender, EventArgs e)
{}
```

Steps to PUBLISH an event

- define a type derived from EventArgs
- define a delegate type that specifies the prototype of the event handler
- define an event based on the delegate type
- define a protected, virtual metod raises the event
- when an event occurs invoke the previous method

Steps to HANDLE an event

- implement an event handler with signature specified by the delegate object of event
- create a delegate object specified for the event. This delegate should refer to the event handler method.
- attach the event handler using += operator

- delegates are MBV objects
- a delegate is a class that holds a reference to the method that is called when an event is fired
- delegates enable the methods to be abstract

```
Delegate int SomeDelegate(string s, boolean b);
Private int SomeFunction(string str, boolean bln)
{}
SomeDelegate sd = new SomeDelegate(SomeFunction);
sd(“abc”,true);
```

Error / Exceptions

Exceptions are unusual occurrences that happen within the logic of an application.

handled exceptions - exceptions handled by try-catch block
unhandled exceptions - exceptions not handled by try-catch
- exceptions not caught by try/catch blocks
- asp.net stops processing a web page after encountering an unhandled exception
- can be trapped at Page lever or Application Level

FCL provides 2 types of exceptions
1. **ApplicationException** - represents exceptions thrown by application
2. SystemException - represents exceptions thrown by CLR
- these classes exist to differentiate exceptions in application from exceptions in CLR.

Different modes of error handling

➢ try-catch-finally

```
try
{        }
catch (FormatException fe)  // special error handlers should be before generic eh
{        }
catch (Exception e)
{        // catch all CLS complaint exceptions        }
catch
{        // catches all other exception including NON-CLS complaint exceptions }
finally
{        // this block is always called
                 // you cannot have more than one finally block for a try block
         // transfer statements like goto, break, continue are not allowed here   }
```

Note:
1. transfer-controls statements such as goto, break, continue, if exist in the try or catch block, transfer will happen only after the finally block execution [if exists].
2. if no matching catch block is found the unhandled exception is propagated to the caller

➢ custom error pages

1. web.config [handling at global level]

```
<customErrors mode="On|Off|RemoteOnly"
defaultRedirect="errGeneric.aspx">
        <error statusCode="403" redirect="errForbiddenPage.htm" />
        <error statusCode="404" redirect="errPageNotFound.htm" />
</customErrors>
```

problem with this redirection is that you will lose the exception information during the redirection process.

2. <%@ Page language="c#" errorPage="specificErrorPage.htm" %> [handling at page level]

➢ error handling at global.asax, application level, page level, page event

when an unhandled exception occurs the following events are fired in successive order

Page.Error	- handled by Page_Error()	[code written in page]
Application.Error	- handled by Application_Error()	[code written in Global.asax]

throw() - to throw/raise an exception,

throw statement is an expensive statement.

custom exceptions
----

1. always derive a custom exception class from System.ApplicationException
2. naming convention used can be myOwnCustomException
3. implement 3 constructors with signatures as given

```
myOwnCustomException()
myOwnCustomException(string msg) : base(string msg)
myOwnCustomException(string msg, Exception inner) : base(string msg, Exception inner)
```

4. important properties of exception class
   - ex.Source
   - ex.Message
   - ex.InnerException
   - ex.StackTrace

```
using(...)
{       // use allocated resources
}
// Dispose() is called automatically for the objects referenced in the parenthesis of the using method
```

The best exception handling is to
----

1. handle the error in the Application.Error event i.e, in the Global.asax page's Application_Error()
2. log the error details
3. mail the error details to admin
4. do the Server.Transfer for HttpUnHandledException to a error display page
5. and let the other errors be handled through web.config <customErrors> tag

note: operations involving floating-points never produce exceptions

error / event logging in code in Application_Error() of GLOBAL.ASAX

```
// this is a better place to log error events
//////////////// event loging ////////////////////////////////
Exception ex = (Exception) Server.GetLastError().InnerException;

Response.Write("<h5>Error Caught in Application_Error event of Global.asax</h5>");
Response.Write(ex.Message);

// create an event source
if (!EventLog.SourceExists("TestJettyEventLog"))
{       EventLog.CreateEventSource("TestJettyEventLog","MyNewLog");
}

// create an event log and assign its source
EventLog el = new EventLog();
el.Source = "TestJettyEventLog";
el.WriteEntry(ex.Message);

// clear the error | so that it doesn't trigger subsequent error events or appear to the user in the browser.
Server.ClearError();
```

Authorization / Authentication / Impersonation

Authentication refers to the process of obtaining credentials from a user and verifying their identity.

The following are the various authentication modes

- **none**
  Users will not be authenticated at all. Pages can simply be delivered to all comers
- **windows** [uses Windows authentication module]
  - *anonymous*
    No authentication, everyone is given access to asp.net application
  - *basic*
    Users must provide windows username/password, info is sent in clear text across network
  - *digest*
    Users must provide windows username/password, but info is hashed and sent across network
  - *integrated*
    Users must provide windows username/password, but info does not travel across network, Kerbos or Challenge/Response protocol is used for user authentication
- **forms** [uses Forms authentication module]
  Uses an HTML form to request credentials from the user. If the credentials are acceptable, the application sends back an identity key. Credentials are then stored in a cookie.
- **passport** [uses Passport authentication module]
  Microsoft's centralized authentication and profile service for member sites.
- **IIS Authentication**
  IIS supports several alternative methods for authentication. These methods interact with the authentication choices within ASP.NET. IIS authentication can also protect resources that are not controlled by ASP.NET.
- **Custom**
  It is very difficult to design a custom secure authentication scheme. This is not recommended for anyone who doesn't have an advanced understanding of cryptography.

Authorization refers to granting rights based on that identity to use various resources.

Impersonation

- <identity impersonate="false"/> Default setting. ASP.NET will always run with its own privileges. All requests will run in the context either the ASPNET account or the system account. This is true whether you're using anonymous access or authenticating users in some fashion.
- <identity impersonate="true"/> ASP.NET takes on the identity passed to it by IIS. If you're allowing anonymous access in IIS, it means that ASP.NET impersonates the IUSR_ComputerName account that IIS itself uses.

Role-based security allows you to authorize access based on user identity or group membership.

IsInRole(), WindowsPrincipal, WindowsIdentity

In Web.config

```
<authentication mode="Forms">
	<forms loginUrl="frmLogin.aspx" name="315C15" timeout="1" />
</authentication>
<authorization>
	<allow users="*" />
	<deny users="?" />
</authorization>
```

Trace / Debugging

Debug and Trace classes in the ***System.Diagnostics*** namespace are tools to help you ferret out errors and unexpected behaviors in your application. These classes let you display alerts or write messages based on results within your application.

Debug methods and properties are automatically stripped out of code compiled for release, whereas, Trace methods and properties are retained in release code

Assert() - The Debug and Trace classes' Assert() tests a value and displays an alert if the value is False. Use Assert() to halt an application for an unexpected result.

Write() / WriteLine() - Use the Write or WriteLine methods to write a message without halting the application. By default, these messages are displayed in the Visual Studio .NET Output window during debugging.

Tracing - is a process of monitoring an executing program
- is a process for collecting information about program execution
- is a technique for recording events, such as exceptions, in an application.
- is the process of displaying execution information

Tracing can be done using 2 different techniques:

- TraceContext calss [***System.Web***]
  - message in page output
  - trace.axd (trace viewer utility)
- Trace & Debug classes [***System.Diagnostics***] - for conditional compilation
  - output window
  - text files
  - event log
  - custom trace listener

TraceContext class is responsible for gathering execution details of a web request.

To record debug and trace messages on a deployed application, create a TextWriterTraceListener class and add it to the Debug or Trace class's Listeners collection.

```
Debug.Listeners.Add(new TextWriterTraceListener("Results.log"));
Debug.WriteLine("Starting tests.");
Debug.Flush();
```

TraceContext
Trace class
Debug class
Trace Listeners
Trace Switches
Conditional Compilation

Listeners are the classes responsible for forwarding, recording and displaying the messages generated by the Trace and Debug classes.

Trace Listeners are objects that receive the tracing information, store it, and then route it to its final destination. Final destination of the tracing information is decided by the Trace Listener. .NET provides following Trace Listeners:

- DefaultTraceListener
- TextWriterTraceListener
- EventLogTraceListener

TraceSwitches are used to set the parameters that control the level of tracing that needs to be done on a program.

TraceSwitches

- BooleanSwitch
- TraceSwitch

Conditional compilation

EventLogs

Debugging

State Management

Asp.Net provides State Management at 4 levels.

1. Application object
2. Session object
3. Request object
4. ViewState thru StateBag

Application

- Static Objects – gets a collection of all instances of all objects declared in global.asax using the <object> scope set application
- All write methods to the Application object implicitly apply a writing lock.
- Use Locks only if you want to shield a group of instruction from concurrent writings
- It is the global repository of data

Session

- Session string is 15 bytes and is made of URL allowed characters only
- Session can be maintained thru COOKIES or MODIFIED URL
- ASP.NET_SessionId is the name of the session cookie name
- ASPFilterSessionId header for cookieless session
- Cookieless sessions cause a redirection when a session starts or when an absolute URL is invoked
- click breaks the session for cookieless session instead use ApplyAppPathModifier()
- <a runat="server" href="<% = Response.ApplyAppPathModifier("/test/abc.aspx")%>>click
- Always use ApplyAppPathModifier() when redirecting from HTTP to HTTPS
- Session dictionary contains object types, to read data back you need to cast the return values   String temp = (String) Session[""MyData]
- Session_OnEnd signals the end of session and is supported only in InProc session mode. For this event to be fired a session has to exist.
- When the first value is added to session dictionary an item is inserted into Cache.
- For InProc session [session held in worker process] the physical container for the session state is cache object.
- State Server session and SQL Server session [out-of-process session] do not work with Cache object.

HTTP Context

- a.l.a REQUEST scope of coldfusion
- this is used to manage information across the entire Request lifetime.

ViewState

- CALL CONTEXT
- _ _VIEWSTATE hidden field
- lets application build a call context and retain values across 2 successive requests for the same page
- StateBag is the class behind the ViewState.
- Every control has a EnableViewState="true|false" attribute
- LosFormatter

Session state is stored at

- In-process storage
  Stored in memory, this is the default for session state storage
- Session state service
  There are two main advantages to using the State Service. First, it is not running in the same process as ASP.NET, so a crash of ASP.NET will not destroy session information. Second, this allows you to share state information across a Web garden (multiple processors on the same computer) or even across a Web farm (multiple servers running the application).
- SQL Server
  Like the State Service, SQL Server lets you share session state among the processors in a Web garden or the servers in a Web farm. But you also get the additional benefit of persistent storage.

Session is maintained through non-persistent cookie SessionID.

Cache

Cache object is a global, application-specific repository of data with advanced capapbilities such as priority, file dependencies, removal call backs, and expiration dates. It periodically checks the memory and if the memory reaches a guard level a scavenging mechanism starts and reclaims memory.

1. Output page caching
2. Cache class

- Cache class works like an application wide repository.
- Cache object is equipped with a scavenging tool.Scavenging tool eliminated low-priority and seldom used cache objects.
- Cache object is created per AppDomain basis
- You can access cache using Http.Cache or Page.Cache
- Cache live side by side with Application object
- Cache[“abc”] = “mydata”; stays in cache indefinetly
- To set expiration policy on a cache use .Insert() or .Add() methods
- Setting a SlidingExpiration > 1 year will throw and error
- Cache Dependency
- By default, all items added to the cache have no expiration date. But .Add() and .Insert() allow you to choose 2 mutually exclusive expiration policies – AbsoluteExpiration -SlidingExpiration
- Cache class is a collection that is instantiated within the HttpRuntime Object

Caching is the technique of storing frequently used items in memory so that they can be accessed more quickly. In other words this refers to storing information for later retrieval.

Types of caching ASP.NET pages

- output caching [for entire page] – caches embedded control too.
- refers to caching the entire output of a page request
- attributes are Duration, Location, VaryByParam, , VaryByHeader, VaryByCustom
- <% @OutputCache Duration="60" VaryByParam="ddCountry;ddState" %>
- <% @OutputCache Duration="60" VaryByHeader="Accept-Language" %>
- <% @OutputCache Duration="60" VaryByCustom="appName" %>
- <% @OutputCache Location="Any|Client|Downstream|None|Server|ServerAndClient" %>

- fragment caching [for partial page]
    - *Fragment* caching refers to caching part of a page. ASP.NET enables fragment caching by allowing you to specify cache rules for Web forms *user controls*. Fragment caching is like output caching, where the output of a user control is cached.
    - VaryByControl is supported only for user controls
    - <% @OutputCache %>

- data caching [application data caching]
    - The caching of arbitrary data is known as *data caching* or *application data caching*. Any arbitary data such as strings, data sets, and so on can be cached. You can cache any object you like in ASP.NET by calling the Add() or Insert() methods of the Cache object. You can also set a sliding expiration time to enable the expiration time to slide whenever the data in the cache is accessed.
    - Cache[“CustomersDataSet”] = dsCustomers;

`Use HttpCachePolicy to set the caching policy.`

The **HttpCachePolicy** class allows you to set output caching for a page programmatically. The Cache property of the HttpResponse object provides access to the HttpCachePolicy class. The HttpResponse object can be accessed through the Response property of the Page or HttpContext class.

HttpCachePolicy represents a programming interface alternative to using the @OutputCache directive

```
private void Page_Load(object sender, System.EventArgs e)
{       lblGenerateTime.Text =DateTime.Now.ToLongTimeString();
        Response.Cache.SetExpires(DateTime.Now.AddSeconds(15));
        Response.Cache.SetCacheability(HttpCacheability.Public);
        Response.Cache.SetValidUntilExpires(true);
}
```

[PartialCaching(5)] – alternative to partial caching using VaryByControl
----
The cache location is determined by the web form setting. Location settings on a user control have no effect.

If the web form's cache duration is longer than the user control's, both the web form response and the user control response will expire using the web form settings.

When the memory is running low, ASP.NET might remove data from the cache if the cached data is not used frequently or is unimportant. This operation is called *scavenging*.

Data Binding

DataBinding referes to the process of making a linking between the UI and Data [data model].

Simple Data Binding - connected a UI element to a single value from the Data Model
- <%# %>
- DataBind()

Complex Data Binding - bind a UI control to an entire collection of Data.

Filtering [select a part of the data from a larger data]
- Filtering with a DataView Object
- Filtering at DataBase Server

Transforming Data - you transform dat using lookups.
- Lookup is a technique for replacing one column data with another in the same table

Template - a set of HTML elements and controls that collectively specify a layout for a control
- Repeater
- DataList
- DataGrid [has a default look and feel]

DataBind() - DataBind() is a method of a Page object and of all server controls

Drag and Dropping

Drag and Dropping a DATABASE creates SqlConnection
Drag and Dropping a TABLE, VIEW creates SqlAdapter
Drag and Dropping a STORED PROCEDURE creates SqlCommand

Repeater

ItemTemplate - Use this template for elements that are rendered once per row of data.
AlternatingItemTemplate - Use this template for elements that are rendered every other row of data. This allows you to alternate background colors.
HeaderTemplate - Use this template for elements that need to be render once before your ItemTemplate section.
FooterTemplate - Use this template for elements that you want to render once after your ItemTemplate section.
SeperatorTemplate - Use this template for elements to render between each row, such as line breaks.

Data Manipulation

1. data stored in relational data bases
2. data stored in flat files
3. data stored in XML documents

T-SQL - ad choc queries
- stored procedures

ADO.NET

Data Provider objects
- Connection - represents a single persistent connection to the database
- Command - represents a query or a stored procedure
- Parameter - represents a single parameter to the stored procedure
- DataReader - fastest way to retrieve a resultset from DB.It is forward-only and read-only. It cannot be modified.
- DataAdapter - provides a bridge between data provider objects and dataset objects

DataSet objects [represents data in abstract form not tied to any particular DB]
- DataSet - is a self-contined, memory-resident representation of relational data, also contains DataTable and DataRelation objects
- DataTable - represents a single table within a DataSet
- DataRow
- DataColumn
- DataRelation
- DataView

Data Provider Classes- connected

SqlConnection[represents a connection to the database]
.Open()
.Close()
.CreateCommand()
.BeginTransaction()
.ConnectionString

SqlCommand [represents a single query that can be sent to the database]
.ExecuteScalar()
.ExecuteReader()
.ExecuteNonQuery() - for no results
.ExecuteXmlReader()
.Parameters

SqlDataReader .Read()
.Close()
.IsDbNull()
.GetXXX()

```
                SqlDataAdapter      [acts as a pipeline between SqlConnection and DataSet]
                                        .Fill()
                                        .Update()
                                        .SelectCommand()
                                        .InsertCommand()
                                        .UpdateCommand()
                                        .DeleteCommand()

DataSet Related Classes     - disconnected
                - DataTable
                - DataRow
                - DataColumn
                - DataView
                - DataRelation

                DataSet         .Tables
                                .Relations
```

Data Error Handling
----
1. SqlException
2. SqlError

```
odbcDSN             =       "DSN=dsnSQLServer;
                             Uid=sa; Pwd=pwd;"
oracleDSN           =       "Data Source=BVRHDB1_DEVRW;
                             User Id=OHDNR; Password=DMANAGER;"
sqlDSN              =       "Data Source=VIRAT; Initial Catalog=bharat;
                             User Id=sa; Password=pwd;"
oleDSN              =       "Provider=SQLOLEDB;
                             Data Source=VIRAT; Initial Catalog=bharat;
                             Uid=sa; Pwd=pwd;"
```

Connection Object

```
//SQL Server Database (Using SQL Server Provider)
String connectString = "server=localhost; uid=sa; pwd=; database=pubs";
SqlConnection myConn = new SqlConnection(connectString);
myConn.Open();

//Jet Database (Using OleDB Provider)
String connectString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=pubs.mdb"
OleDbConnection myConn = new OleDbConnection(connectString);
myConn.Open();

//Jet Database when MDB is in a different folder (Using OleDB Provider)
String connectString = "Provider=Microsoft.Jet.OleDb.4.0; Data Source=" +
Server.MapPath ("~/shared/data/dbtutor.mdb");
OleDbConnection myConn = new OleDbConnection(connectString);
myConn.Open();
```

Command Object

```
//Creating the Command Object (Using SQL Server Provider)
String query = "SELECT * FROM Authors";
SqlCommand myCmd = new SqlCommand(query, myConn);

//Creating the Command Object (Using OleDB Provider)
String query = "SELECT * FROM Authors";
OleDbCommand myCmd = new OleDbCommand(query, myConn);
```

Reading Data with DataReader

```
//Reading the Data with DataReader (Using SQL Server Provider)
SqlDataReader myReader = myCmd.ExecuteReader ();

while (myReader.Read())
{	Response.Write (myReader["LastName"] + ", " + myReader["FirstName"]);
}

myReader.Close();
myConn.Close();

// Reading the Data with DataReader (Using OleDB Provider)
OleDbDataReader myReader = myCmd.ExecuteReader();

while (myReader.Read())
{	Response.Write (myReader["LastName"] + ", " + myReader["FirstName"]);
}

myReader.Close();
myConn.Close();
```

Reading Data with DataAdapter/DataSet

```
//Using DataAdapter instead of DataReader (Using SQL Server Provider)
SqlConnection myConn = new SqlConnection("server=(local)\\NetSDK;
trusted_connection=yes; database=northwind");

SqlDataAdapter myCmd = new SqlDataAdapter("select * from customers", myConn);

DataSet myData = new DataSet();

myCmd.Fill (myData, "Customers");

foreach (DataTable table in myData.Tables)
{	foreach (DataRow row in table.Rows)
	{	foreach (DataColumn col in table.Columns)
		{	Response.Write (row[col].ToString());
		}
	}
}
```

```
//Using DataAdapter/DataSet instead of DataReader (Using OleDB Provider)
OleDbConnection myConn = new OleDbConnection("Provider=Microsoft.Jet.OleDb.4.0; Data
Source="Server.MapPath("~/shared/data/dbtutor.mdb"));

OleDbDataAdapter myCmd = new OleDbDataAdapter("select * from Products", myConn);

DataSet myData = new DataSet();

myCmd.Fill (myData, "Products");

foreach (DataTable table in myData.Tables)
{	foreach (DataRow row in table.Rows)
	{	foreach (DataColumn col in table.Columns)
		{	Response.Write (row[col].ToString());
		}
	}
}
```

Using DataSet to Read XML File

```
DataSet myData = new DataSet();
myData.ReadXml(Server.MapPath ("~/adonet/demos/authors.xml"));

foreach (DataTable table in myData.Tables)
{	foreach (DataRow row in table.Rows)
	{	foreach (DataColumn col in table.Columns)
		{	Response.Write (row[col].ToString());
		}
	}
}
```

Deployment

xcopy - to depoly an assembly when the assembly is used by only one application and requires no additional setup steps to occur on the target computer

ftp -

windows installer [.msi]

- setup project
- web setup project
- merge module project
- cab project

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/gngrfnetframeworkconfigurationfileschema.asp

File

There are 2 ways to access files

1. File - represents a disk file
   - contains static methods to open, move, copy and delete files,
   - performs security check whenever a static method is called which involves additional overhead
   - also helps in creating FileStream objects

2. FileInfo - almost everything can of File class actions can be carried out
   - does not contain static methods
   - need to create and instance of this class before using it
   - security check is performed only when the object is created and is not repeated for each method call.

Directory (static) / DirectoryInfo (instance)

Streams - a flow of raw data
Backing Stores - represents a place to put the data

TextReader [abstract]
    StreamReader
    StringReader

    BinaryReader

TextWriter [abstract]
    StreamWriter
    StringWriter

    BinaryWriter

Stream [abstract]
    FileStream
    MemoryStream
    NetworkStream
    CryptoStream
    BufferedStream

FileStream

This class treats file as raw, typeless stream of bytes.
This is the best option if you do not know the internal structure of the file.

StreamReader / StreamWriter
----
best for .txt [TEXT] files

BinaryReader / BinaryWriter

Globalization

Globalization is the process of creating an application that meets the needs of users from multiple cultures.

Localization is accommodating the cultural differences in an application.

Localization is a 3 step process consisting of
- globalization [identifying resources like text, dates, times, currency etc]
- localizability [verifying seperation of resources from code]
- localization [translating resources]

Culture

Cultures are indentified by culture codes and are of 3 types
- neutral culture code [specify general languages such as English, Spanish]
- specific culture code
- invariant culture

en - neutral culture
- does not specify subculture code
- does not provide sufficient information to localize an application

en-GB - specific culture
- provides enought inofrmation to localize an application

az-AZ-Cyrl - specific culture with 2 subculture codes

Invariant culture - does not have abbreviation
- has 2 purposes
- to interact with other software such as system services
- to store data in a culture independent format

System.Globalization.CultureInfo

CultureInfo.GetCultures() is a static method

.NET handles localization on a thread-by-thread basis.
- Thread.CurrentThread
- CurrentCulture - dictates formats for dates, currency, numbers, casing rules and string comparision rules.
- CurrentUICulture - for choosing resources for UI

Request.UserLanguages[0] gives the client browser culture info

english	name	age	city	save
french	nom	âge	ville	économiser
spahish	nombre	edad	ciudad	excepto

Resources01.resx - invariant culture resource file
Resources01.es-MX.resx - specific culture resource file
Resource01.es.resx - neutral culture resource file

mirroring - using <html dir=rtl>

string indexing

.NET uses the Request object’s UserLanguages property to return a list of the user’s language preferences.

To get the user’s culture at run time:
1. Get the Request object’s UserLanguages property.
2. Use the returned value with the CultureInfo class to create an object representing the user’s current culture.

Globalization approaches
1. Detect and Redirect
2. Run-Time adjustments
3. Satellite assemblies

Setting culture in web.config
<globalization requestEncoding="utf-8" responseEncoding="utf-8" culture="ar-SA" />

mirroring
<body dir="rtl">

Threads / run-time / culture
you can set the culture dynamically at run time using the Thread class’s CurrentCulture property,

The CurrentCulture property can’t be set to a neutral culture because it uses region information to format currencies, among other things.

Encoding
The common language runtime (CLR) uses the UTF-16 character encoding.
ASP.NET uses the UTF-8 character encoding by default when interpreting requests and composing responses.

```
<globalization requestEncoding="utf-8" responseEncoding="utf-8"
                                        fileEncoding="utf-8" />
```

Help

Tool Tips

A ToolTip is a short, descriptive message that’s displayed when the user places the mouse pointer over a control and leaves it there for a couple of seconds.

Only Internet Explorer displays the title attributes as ToolTips. Other browsers do not display ToolTips.

Displaying Help as Web Forms or HTML

```
<a href="#" onclick="window.open('helpTopic1.aspx','helpwin').focus()">click here
                              for more help</a>
```

Displaying HTML Help / Compiled Help Files

```
<a href="#" onclick="window.showHelp('helpTopic1.aspx')">click here for showHelp</a>
```

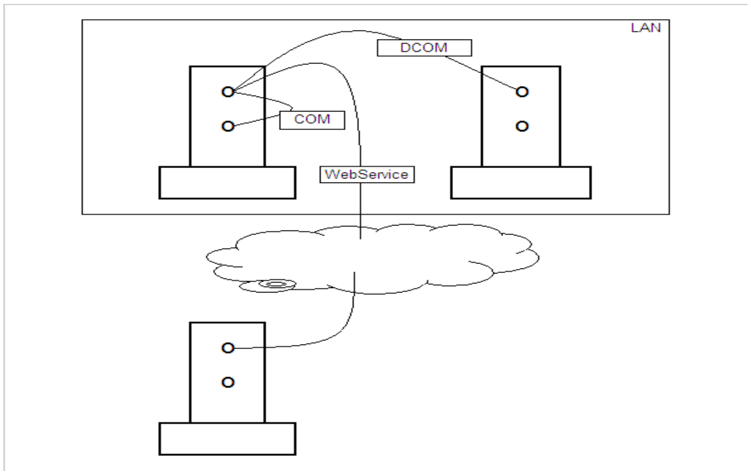
Testing

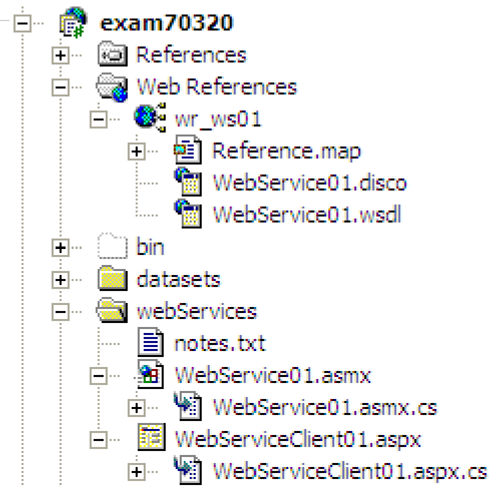
testing is the process of executing a program with the intention of finding errors

correctness / robustness / reliability

incremental testing / evolutionary testing
- unit testing [performing basic tests at component level]
- integration testing
    - bottom-up approach
    - top-down approach
    - umbrella approach
- regression testing
-

Web Services





Step 1 - Search UDDI or any other DIRECTORY service to find an XML Web Service [if you do not know the location of the web service] This server returns the URL link to the web services .DISCO file [DISCOVERY FILE] http://www.contoso.com/default.disco

Step 2 - Contact Contoso.com server for the .DISCO file.

Step 3 - The received .DISCO file is analyzed. The client uses the info in the discovery document and requests a server to return the SERVICE DESCRIPTION [.WSDL file] of the XML web service.

- To manually create web services files
  - Web Services Discovery Tool
    C:\> disco http://live.capescience.com/wsdl/AirportWeather.wsdl command
    Creates 2 files AirportWeather.wsdl
    Results.discomap
  - Web Services Description Language Tool
    C:\> wsdl /language:CS /out:AirportWeatherProxy.cs AirportWeather.wsdl
    Creates a proxy file called AirportWeatherProxy.cs

DISCO [Discovery of Web Services | directory, inspection, description]
- is the process that clients use to locate documents that describe WS using WSDL.
- to find a web service and
- to discover a web service details.
- Encapsulates oo messages between web service servers and clients

SOAP [Simple Object Access Protocol]
- SOAP protocol is used for exchanging STRUCTURED and TYPED info between the application and internet.
- a way to translate Objects [methods + properties] into XML
- a way to encapsulate method calls as XML sent via HTTP
- tells how to package the data for transport

UDDI [Universal Description, Discovery and Integration]
- Directories provide a central place to store published information about WS.
- UDDI defines the guidelines for the publishing
- is like a phone book [combination of white and yellow pages].
- This protocol enables you to find web services by connecting to a directory

    - Is the process of finding services by referring to a central directory
    - UDDI Registries come in 2 forms PUBLIC and PRIVATE

WSDL [Web Services Description Language],

    - provides info that enables you to know which operations to perform on a WS.
    - Specifies the SOAP message schemas.
    - This document mentions the SOAP message schemas [methods info]
    - defines the public interface of a web service
    - WSDL lets you retrieve information on the classes and methods that are supported by a particular web service
    - You can create PROXY CLASS using wsdl.exe
    - is a standard by which a web server tells it clients
        - what messages it expects and
        - which results it will return

Web Method

- [WebMethod (CacheDuration, EnableSession, TransactionOption)]

Web Reference

- When a web reference is set in VS.NET the software handles the details of discovery automatically for you.
- You can automatically generate proxy classes for a Web service by setting a Web reference to point to the Web service.

- Use ASP.NET to create a simple Web service.
  Then use the wsdl.exe tool to create a proxy class for that Web service,
  Lastly, instantiate the Web service within another ASP.NET application.
- When you create the Web reference, for example, Visual Studio .NET reads the appropriate WSDL file to determine which classes and methods are available from the remote server. When you call a method on an object from that server, the .NET infrastructure translates your call and the results into SOAP messages and transmits them without any intervention on your part.
- [WebMethod CacheDuration=60 EnableSession=true Description=”” TransactionOption=TransactionOption.Required]
- The only time you want to set the BufferResponse=True is when you are returning more than 16KB of data, and that the data is created on the fly.
- Discovery is the process of finding the web services and determining their interfaces.
- Web Services SERVERS and CLIENTS communicate through XML messages transmitted over HTTP.

Configuration

If you change the <processModel> element in the machine.config file, you must restart IIS to have the changes take effect.

<location path="subDir1">
settings contained in this element will apply only to pages stored in the subDir1 subdirectory of the application
</location>

allowOverride="false" attributes
- more speciifc config. files cannot override this setting

Configuration file is an xml document.
- Machine.config
- Application.config
- Security.config [mscorcfg.msc | caspol.exe]

check site www.hunterstone.com for configuration editor

A Matter Of Context by: dotScott

For passing values between asp.net forms in C#, there are two ways to do this besides the traditional request, application, session objects that were used in ASP. There is the Context.Items collection and the Context.Handler Context.Items is available throughout any of the pages within the given context (like the session object).

The syntax is simple for adding:

```
Context.Items.Add("userName", name.Text)
```

then retrieving the value is:

```
Context.Items("userName")
```

For the Context.Handler - this only allows passing values between two pages (not more than two- where context.items allows passing values through more than two pages) in the calling page (CallingPage)

```
private void Button1_Click(object sender, EventArgs e){
Server.Transfer("CalledPage.aspx");
}
```

Then in the called page:

```
CallingPage sourcepage = (CallingPage) System.Web.HttpContext.Current.Handler;
Label1.Text = sourcepage.Property1.Text;
```

The Context object is initialized at the start of each request and will last until the end of that request. So Context object provides the ease of getting the data from one page to another.

Context is an object of type System.Web.HttpContext. It is exposed as a property of the ASP.NET Page class. It's also available from user controls and your business objects (more on that later). Here's a partial list of the objects rolled up by HttpContext:

Object	Description
Application	A key/value pair collection of values that is accessible by every user of the application. Application is of type System.Web.HttpApplicationState.
ApplicationInstance	The actual running application, which exposes some request processing events. These events are handled in Global.asax, or an HttpHandler or HttpModule.
Cache	The ASP.NET Cache object, which provides programmatic access to the cache. Rob Howard's ASP.NET Caching column provides a good introduction to caching.
Error	The first error (if any) encountered while processing the page. See Rob's Exception to the Rule, Part 1 for more information.
Items	A key-value pair collection that you can use to pass information between all of the components that participate in the processing of a single request. Items is of type System.Collections.IDictionary.
Request	Information about the HTTP request, including browser information, cookies, and values passed in a form or on the query string. Request is of type System.Web.HttpRequest.
Response	Settings and content for creating the HTTP response. Request is of type System.Web.HttpResponse.
Server	Server is a utility class with several useful helper methods, including Server.Execute(), Server.MapPath(), and Server.HtmlEncode(). Server is an object of type System.Web.HttpServerUtility.
Session	A key/value pair collection of values that are accessible by a single user of the application. Application is of type System.Web.HttpSessionState.
Trace	The ASP.NET Trace object, which provides access to tracing functionality. See Rob's tracing article for more information.
User	The security context of the current user, if authenticated. Context.User.Identity is the user's name. User is an object of type System.Security.Principal.IPrincipal.

Passing Values Between Web Forms Pages by Scott Brookhart

- Reponse.Redirect issues an HTTP 304 to the browser and causes the browser to go to that page."Response.Redirect() tells the browser that the page requested has moved to a new location. The browser then issues a request for the second page

  Response.Redirect introduces some additional headaches. First, it prevents good encapsulation of code. Second, you lose access to all of the properties in the Request object. Sure, there are workarounds, but they're difficult.

  Finally, Response.Redirect necessitates a round trip to the client, which, on high-volume sites, causes scalability problems. As you might suspect, Server.Transfer fixes all of these problems. It does this by performing the transfer on the server without requiring a roundtrip to the client.

- Server.Transfer starts executing another page without sending anything to the browser.Server.Transfer() is all server side. Effectively what happens is when ASP gets to Server.Transfer() command, it takes all the code in "somepage.asp" and pastes it onto the bottom of the current page, and processes the new page."

  Client is shown as it is on the requesting page only, but the all the content is of the requested page. Data can be persist across the pages using Context.Item collection, which is one of the best way to transfer data from one page to another keeping the page state alive.

  Client knows the physical location (page name and query string as well). Context.Items loses the persistence when navigate to destination page. In earlier versions of IIS, if we wanted to send a user to a new Web page, the only option we had was Response.Redirect. While this method does accomplish our goal, it has several important drawbacks. The biggest problem is that this method causes each page to be treated as a separate transaction. Besides making it difficult to maintain your transactional integrity.

Remoting

- .NET Remoting allows communication between 2 objects (which can be on different OS and on 2 different computers)
- Communication channels are the objects that transport messages between the remote objects. Transport channel is the combination of the following technologies A] opening a network connection B] formatting messages C] streaming messages.
- Channels are categorized as
  Receiver / Server ←-→ Sender / Client
- Formatters – formatters code/decode | serialize/deserialise (marshalling) messages
  - Binary
  - SOAP (XML)
- Transportation protocols are
  - HTTP
  - TCP/IP
- Object lifetime leases
- .NET Remoting uses proxy objects to allow the use of server object in the client process. Proxy object contains references to all the methods and properties of the server object.
- REMOTABLE OBJECT (2 types of remoting objects)
  - Marshal By Value
    - Should implement ISerializable interface
    - OR should be marked by SerializableAttribute
    - .NET creates a copy of entire server object and send it over to the client application/process.
    - Reduces network trips
    - Access is fast as marshalling does not take place
  - Marshal By Reference
    - Should extend System.MarshalByRefObject
    - .NET creates a proxy object in the client AppDomain that represents the server object and return a reference of that proxy to the caller/client
    - Only MBR can be actuated remotely
- REMOTING OBJECT
  - Server Activated (SA objects are created when a server method is called)
    - Singleton
    - SingleCall
  - Client Activated (CA objects are created using the NEW keyword)
- When you create an instance of the remote object in client application, .NET creates a proxy object and sends it to the client application/process. This proxy object contains references to all methods and properties of the server object.
- Lifetime leases – lifetime of an object is the duration for which the object resides in memory
- Lease manager
- Marshalling
  - When a client calls a remote object the .NET creates a message that contains parameters and other call related info. This way of bundling the info into a message is called marshalling.
- Message sink is an object that allows client to establish a connection with the channel registered by the remote object and forward the messages to the channel.
- Channel Sinks
  - Formatter Sinks
  - Transport Sinks
- Sponsors

- .NET Remoting – when both end points (client + server) of a distributed application are in our control
- ASP.NET webservices – when one end point (server) is in out control
- .NET Remoting enables objects in different AppDomains to talk to each other.
- Channels transport messages across remoting boundaries such as appDomains, process and computers
- The .NET REMOTING API is used for accessing an object in an external AppDomain.
- Remoting Objetcs
    - Remoting Class
    - Remoting Host
        - Console Application
        - Window Service
        - IIS
    - Remoting Client
    - Channels
    - Formatters
    - Activation modes
- To access the code in another AppDomain and application needs to use a proxy. A proxy is an object that allows interprocess communication.

Serviced Components

- Windows DNA (Windows Distributed internet Application) enables implementation of a 3-tier application architecture, which allows you to maintain business logic and data access code in components called business objects.
- PRESENTATION SERVICES   APPLICATION SERVICES   DATA SERVICES

    SYSTEM SERVICES
- Transactions group a set of tasks into a single execution unit.
- COM+ components automatically participate in transactions

    BeginTransaction → CommitTransaction
    or
    AbortTransaction (RollbackTransaction)
- ACID (Atomicity, Consistency, Isolation, durability)
- COM+ component exists in 3 states; exists and activated; exists and not activated; nonexistent;
- System.EnterpriseServices
- After you created a serviced component, you need to host it in a COM+ application. For that you must assign a strong name to the assembly, register the assembly in the windows registry and install the type library definition in a COM+ application.
- COM+ application is like AppDomain
- Tlbexp.exe

XML

- DOM (Document Object Modle) is a standard for representing information as a tree of nodes contained in a HTML/XML document.
- XmlReader provides forward only, read only access to xml file
- XmlDocument object represents an entire xml document
- XmlNode represents a single node in DOM.
- XmlDataDocument allows connections between XmlDocument & DataSets.
- XPath is like a query language
- XPath starts with the notation of current context
  ./ uses current node as current context
  / uses the root of xml document as current context
  .// uses the entire xml hierarchy starting with the current node as current context
  // uses the entire xml document as current context
- XpathNavigator allows random-access navigation of xml file
- prime use of schema file (.xsd) is to validate the corresponding xml file
- 4 ways of getting .xsd file (xml schema)
  - generate file by SQL Server/MSAccess
  - create own schema files from scratch
  - DataSet.ReadXmlSchema()
  - DataSet.InferXmlSchema()
- A valid xml file is one which conforms to a specific DTD or SCHEMA
  DiffGram – think this as a before-and-after snapshot of a part of a SQL Server Table.
  DiffGrams can insert or delete or modify data.
- FOR XML clause in the SQL is used to directly generate XML from SQL Server
  FOR XML AUTO
  RAW
  EXPLICIT
  AUTO, XMLDATA

Windows Services

- Windows services run as background processes.
- Windows service architecture has 3 components
  - Service application
  - Service controller application
  - Service control manager
- A window service is installed in the registry as an executable object
- SCM manages all window services. SCM is a RPC (remote procedure call) server and supports local/remote management of a service.
- System.ServiceProcess / ServiceBase / ServiceInstaller / ServiceProcessInstaller / ServiceController
- Win32OwnProcess Service – 1 service per process
  Win32ShareProcess service – n service per process