ASP .NET MVC 5

Nemanja Kojic, MScEE
What is MVC?

• Model-View-Controller (MVC)
• Standard Architectural Pattern
• Separation of concerns: model, view, controller
ASP .NET MVC Framework

• An alternative to ASP .NET Web Forms
• Presentation framework
  – Lightweight
  – Highly testable
  – Integrated with the existing ASP .NET features:
    • Master pages
    • Membership-Based Authentication
    • ...
ASP .NET MVC Framework
Components

• Models
  – Business/domain logic
  – Model objects, retrieve and store model state in a persistent storage (database).

• Views
  – Display application’s UI
  – UI created from the model data

• Controllers
  – Handle user input and interaction
  – Work with model
  – Select a view for rendering UI
When to use MVC approach?

• Advantages:
  – Easier to manage complexity (divide and conquer)
  – It does not use server forms and view state
  – Front Controller pattern (rich routing)
  – Better support for test-driven development
  – Ideal for distributed and large teams
  – High degree of control over the application behavior
ASP .NET MVC Features

• Separation of application tasks
  – Input logic, business logic, UI logic

• Support for test-driven development
  – Unit testing
  – No need to start app server

• Extensible and pluggable framework
  – Components easily replaceable or customized
    (view engine, URL routing, data serialization,...)
ASP .NET MVC Features (cont.)

• Support for Dependency Injection (DI)
  – Injecting objects into a class
  – Class doesn’t need to create objects itself

• Support for Inversion of Control (IOC)
  – If an object requires another object, the first should get the second from an outside source (configuration file)
ASP .NET MVC Features (cont.)

• Extensive support for ASP .NET routing
• Building apps with comprehensible and searchable URLs
• Customizable URLs
  – Adapted to work well with search engines
  – Adapted to REST addressing
  – Decoupled from resource files
• Use of existing ASP .NET features (backward compatibility)
ASP .NET MVC App Structure

• URLs mapped to controller classes
• Controller
  – handles requests,
  – executes appropriate logic and
  – calls a View to generate HTML response
• URL routing
  – ASP .NET routing engine (flexible mapping)
  – Support for defining customized routing rules
  – Automatic passing/parsing of parameters
ASP .NET App Structure

• No Postback interaction!
• All user interactions routed to a controller
• No view state and page lifecycle events
MVC App Execution

• Entry points to MVC:
  – UrlRoutingModule and MvcRouteHandler

• Request handling:
  – Select appropriate controller
  – Obtain a specific controller instance
  – Call the controller’s Execute method
MVC App Execution - stages

- Receive first request for the application
  - Populating RouteTable
- Perform routing
- Create MVC Request handler
- Create controller
- Execute controller
- Invoke action
- Execute result
  - ViewResult, RedirectToRouteResult, ContentResult, FileResult, JsonResult, RedirectToResult
MVC App Execution
BUILDING VIEW PAGES USING RAZOR LANGUAGE

RAZOR ENGINE
Razor Engine

- A new view-engine
- Optimized around HTML generation
- Code-focused templating approach
Razor Engine – Design goals

• Compact, Expressive and Fluid
• Easy to learn
• It is not a new language
• Works with any text editor
• Great Intellisense
• Unit-testable
  – Testing views without server, controllers...
Razor – HelloWorld

- Uses @ for Razor blocks

Razor Example
Hello Scott, the year is 2010
Checkout this product

Code Nugget Example with .ASPX file
Hello <%=name %>, the year is <%= DateTime.Now.Year %>
Checkout <a href="/Products/Details/<%=productId %>">this product</a>

Razor file

<razor file>
<h1>Razor Example</h1>
<h3>Hello <%=name %>, the year is <%= DateTime.Now.Year %>
</h3>
<p>Checkout <a href="/Products/Details/<%=productId %>">this product</a>
</p>
</razor file>

.aspx file

<razor file>
<h1>Code Nugget Example with .ASPX file</h1>
<h3>Hello <%=name %>, the year is <%= DateTime.Now.Year %>
</h3>
<p>Checkout <a href="/Products/Details/<%=productId %>">this product</a>
</p>
</razor file>
Loops and Nested HTML

Razor syntax

```
<ul id="products">
    @foreach(var p in products) {
        <li>@p.Name ($@p.Price)</li>
    }
</ul>
```

.aspx syntax

```
<ul id="products">
    @foreach(var p in products) {
        <li>@p.Name (@p.Price)</li>
    }
</ul>
```
If Blocks and Multi-line Statements

IF statement

```
if (products.Count == 0) {
    <p>Sorry - no products in this category</p>
} else {
    <p>We have a products for you!</p>
}
```

Multi-line statement

```
int number = 1;
string message = "Number is " + number;
```

Multi-Token statement

```
<p>Your Message: @("Number is: " + number)
```

Variables can span multiple server code blocks!
Integrating Content and Code

Parser examines right-hand side of @ character.

Identifying nested content with HTML block tag.
Layout/Master page

SiteLayout.cshtml

Including specific body content.
Explicitly setting LayoutPage property.

```csharp
LayoutPage = "SiteLayout.cshtml";
```

Complete HTML page.

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Site</title>
  </head>
  <body>
    
    <div id="header">
      <a href="/">Home</a>
      <a href="/About">About</a>
    </div>

    <div id="body">
      
      <h1>About This Site</h1>

      <p>
        This is some content that will make up the "about" page of our web-site. We'll use this in conjunction with a layout template. The content you are seeing here comes from the Home.cshtml file.
      </p>

      <p>
        And obviously I can have code in here too. Here is the current date/time: @DateTime.Now
      </p>

      <p>
        This is some content that will make up the "about" page of our web-site. We'll use this in conjunction with a layout template. The content you are seeing here comes from the Home.cshtml file.
      </p>

      <p>
        And obviously I can have code in here too. Here is the current date/time: 7/2/2010 2:53:24 PM
      </p>

    </div>

  </body>
</html>
```
Master page – section overrides

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Site</title>
  </head>
  <body>
    <div id="header">
      <a href="/">Home</a>
      <a href="/About">About</a>
    </div>
    <div id="left-menu">
      @RenderSection("menu", optional:true)
    </div>
    <div id="body">
      @RenderBody()
    </div>
    <div id="footer">
      @RenderSection("footer", optional:true)
    </div>
  </body>
</html>
```

This section is optional.

This section is optional.
Master page – section overrides

```html
<!DOCTYPE html>
<html>
<head>
  <title>Simple Site</title>
</head>
<body>
  
  <div id="header">
    <a href="/">Home</a>
    <a href="/About">About</a>
  </div>

  <div id="left-menu">
    @RenderSection("menu", optional:true)
  </div>

  <div id="body">
    @RenderBody()
  </div>

  <div id="footer">
    @RenderSection("footer", optional:true)
  </div>

</body>
</html>
```

```html
<h1>About This Site</h1>

<p>This is some content that will make up the "about" page of our web-site. We'll use this in conjunction with a layout template. The content you are seeing here comes from the Home.cshtml file.</p>

<p>And obviously I can have code in here too. Here is the current date/time: @DateTime.Now</p>

@section menu {
  <ul id="sub-menu">
    <li>About Item 1</li>
    <li>About Item 2</li>
  </ul>
}

@section footer {
  <p>This is my custom footer for Home</p>
}
```
<!DOCTYPE html>
<html>
  <head>
    <title>Simple Site</title>
  </head>
  <body>
    <div id="header">
      <a href="/">Home</a>
      <a href="/About">About</a>
    </div>
    <div id="left-menu">
      <ul id="sub-menu">
        <li>About Item 1</li>
        <li>About Item 2</li>
      </ul>
    </div>
    <div id="body">
      <h1>About This Site</h1>
      <p>This is some content that will make up the "about" page of our web-site. We'll use this in conjunction with a layout template. The content you are seeing here comes from the Home.oshtml file.</p>
      <p>And obviously I can have code in here too. Here is the current date/time: 7/2/2010 3:34:05 PM</p>
    </div>
    <div id="footer">
      <p>This is my custom footer for Home</p>
    </div>
  </body>
</html>
Re-usable “HTML Helpers”

- Methods that can be invoked within code-blocks
- Encapsulate generating HTML
- Implemented using pure code
- Work with Razor engine

Built-in HTML helper

```html
<fieldset>
  <legend>Edit Product</legend>
  <div>
    @Html.LabelFor(m => m.ProductID)
  </div>
  <div>
    @Html.TextBoxFor(m => m.ProductID)
    @Html.ValidationMessageFor(m => m.ProductID)
  </div>
</fieldset>
```
Define own HTML helpers

@helper declarative syntax

Helper’s parameters (full language and debugging support)

HTML Helper definition

HTML Helper should be placed to Views\Helper directory.

HTML Helper Invocation

@helper ProductListing(List<Product> products) {
    <ul id="products">
        @foreach(var p in products) {
            <li>@p.Name ($p.Price)</li>
        }
    </ul>
}

<body>
    <h1>Here are My Products</h1>
    <div>@ProductListing(Model.Products)</div>
</body>
Visual Studio support

```html
<h1>Products</h1>

<ul id="products">
    @foreach (var p in Model.Products) {
        <li>@p.Name</li>
    }
</ul>
```
Razor – Summary

- A good new view engine
- Code-focused templating
- Fast and expressive
- Compact syntax
- Integrated with C# and VB
CREATING ASP .NET MVC APPLICATION
New Project...
New Project

Installed

Templates
- Visual Basic
- Visual C#
- Windows
- Web
  - Cloud
  - Silverlight
  - Test
  - WCF
- Other Project Types
- Samples

Online

.NET Framework 4.5
Sort by: Default

ASP.NET Web Application
Visual C#

Click here to go online and find templates.

Name: MvcMovie
Location: c:\users\riande\documents\visual studio 2013\Projects
Solution name: MvcMovie
Select the project template
ASP .NET MVC App Home page

Your ASP.NET application

URL: C:\tf\MVC5\RTM\MvcMovie\MvcMovie\Project_Readme.html

Congratulations! You’ve created a project

This application consists of:

- Sample pages
- Sample pages showing basic nav between Home, About, and Contact
- Theming using...

CUSTOMIZE APP
- Get started with ASP.NET MVC
- Change the site’s theme
- Add more libraries using NuGet
- Configure authentication
- Customize information about the website

DEPLOY
- Ensure your app is ready for production
- Windows Azure
- Hosting providers

GET HELP
- Get help
- Get more templates

Ready
Run the application...

ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

Getting started

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and gives you full control over markup for enjoyable, agile development.

Get more libraries
Expand the default App menu
ADDING CONTROLLER
Adding controller
Adding controller (cont.)
Adding a controller (cont.)
Testing the controller

```csharp
using System.Web;
using System.Web.Mvc;

namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        //
        // GET: /HelloWorld/
        
        public string Index()
        {
            return "This is my <b>default</b> action...";
        }

        //
        // GET: /HelloWorld/Welcome/
        
        public string Welcome()
        {
            return "This is the Welcome action method...";
        }
    }
}
```

This is my **default** action...
Mapping controller

• Controller selection based on URL
• Default URL routing logic:
  /[Controller]/[ActionName]/[Parameters]
• Format for routing in
  App_Start/RouteConfig.cs

```csharp
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{pathInfo}");

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
    );
}
```
URL routing

• Webapp URL without URL segments => HomeController::Index()
• Index() – default method of a controller
• /HelloWorld => HelloWorldController
• /HelloWorld/Index => HelloWorldController::Index()
• http://webapp:port/HelloWorld/Welcome => HelloWorldController::Welcome()
Parameters

- `/HelloWorld/Welcome?name=Scott&numtimes=4`
- Introducing 2 parameters to Welcome method
- Parameters passed as query strings!

```csharp
public string Welcome(string name, int numTimes = 1) {
    return HtmlUtility.HtmlEncode("Hello " + name + ", NumTimes is: " + numTimes);}
```

Hello Scott, NumTimes is: 4
URL Parameters

- http://webapp/HelloWorld/Welcome/3?name=Rick

```csharp
public string Welcome(string name, int ID = 1)
{
    return HtmlUtility.HtmlEncode("Hello " + name + ", ID: " + ID);
}
```

Parameter ID matches URL specification in RegisterRoutes method.

```csharp
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{pathInfo}");

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
    );
}
```
Views

- Views created using Razor view engine
- Controller method returns View object
- Controller method return type is ActionResult
- Common pattern: all view pages share the same master layout page

```csharp
public ActionResult Index()
{
    return View();
}
```
Create View page
Create View page

Master page.
Implementing View page

@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
<p>Hello from our View Template!</p>

public ActionResult Index()
{
    return View();
}

Selected master page.

Index, by default.

Change controller’s method signature.
The method returns a view object:
searches a view file that is named the same as the method (Index.cshtml).
ViewBag

- Pass data between view template and layout view file
- ViewBag is a dynamic object (has no defined properties)
Passing data from Controller to View

• View is used for data presentation
• Controller must provide a view with the data
• One approach: using ViewBag
  – Controller puts data to ViewBag,
  – View reads ViewBag and renders the data
  – No data binding!
• Alternative approach: the view model
  – Strongly typed approach
Passing data from Controller to View

Controller

```csharp
using System.Web;
using System.Web.Mvc;

namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult Welcome(string name, int numTimes = 1)
        {
            ViewBag.Message = "Hello " + name;
            ViewBag.NumTimes = numTimes;

            return View();
        }
    }
}
```

View

```html
@{
    ViewBag.Title = "Welcome";
}
<h2>Welcome</h2>
<ul>
    @for (int i = 0; i < ViewBag.NumTimes; i++)
    {
        <li>@ViewBag.Message</li>
    }
</ul>
```

Returns HelloWorldView object.
ADDING A MODEL
Model components

• Entity framework - data access technology
• “Code first” development paradigm (first code classes, then generate DB schema)
• “Database first” development paradigm define db schema first, then generate models, controllers and views
Adding a model class

Enter the class name, e.g. Movie.cs
Adding properties to a model class

```csharp
using System;

namespace MvcMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }
}
```
Adding a DbContext class

```csharp
using System;
using System.Data.Entity;

namespace MvcMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }

    public class MovieDBContext : DbContext
    {
        public DbSet<Movie> Movies { get; set; }
    }
}
```
DB Connection string

Separate connection string for each DbContext class
Accessing Model from a Controller
Accessing Model from a Controller

Visual Studio Creates:
A controller MoviesController.cs file in Controllers folder,
Create.cshtml, Delete.cshtml,
Details.cshtml, Index.cshtml in Views\Movies folder.

Strongly typed approach.
Run Application...

Notice: default routing

Creates a new movie.

Database is still empty.

Notice: generic column name, derived from the model class.
Creating a model object

Automatically generated form, based on the model info.
Generated Controller class

```csharp
public class MoviesController : Controller
{
    private MovieDBContext db = new MovieDBContext();

    // GET: /Movies/
    public ActionResult Index()
    {
        return View(db.Movies.ToList());
    }
}
```

- **Instantiated DbContext instance.**
- **Index method.**
Strongly typed models

- MVC provides strongly typed way of passing data from Controller to View
- Better compile-time checking
- Richer IntelliSense in VS code editor
Strongly typed models

@model: Specifies class of the model

Id parameter generally passed as a part of the route.

Communicates with the master page.

Context-sensitive data access.
Model object is strongly typed. Each item is a Movie object.
Edit View

Localhost:1234/movies/Edit/4

URL generated using Html Helpers!
Edit View (cont.)

Parameter passed through the URL query. Works for MVC default URL mapping.

Label defined in the model class.

Date format defined in the model class.
Edit View

Generates hidden anti-forgery token.
Generates html label.
Generates text box.
Generates validation message.
Property annotations

```
using System;
using System.ComponentModel.DataAnnotationsAnnotations;
using System.Data.Entity;

namespace MvcMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }
        [Display(Name = "Release Date")]
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }

    public class MovieDBContext : DbContext
    {
        public DbSet<Movie> Movies { get; set; }
    }
}
```
ActionLink helper

- **Html.ActionLink** – generates a link according to a given URL mapping policy

**Primer:**

```csharp
Html.ActionLink("Edit", "Edit", new { id = item.ID })
```

**Anonymous object** – specifies ID of an object

**Controller action name.**
Edit actions

• Implemented as Controller’s operations

HTTP GET operation

HTTP POST operation

[HttpGet] annotation by default.

[HttpPost] attribute – a security mechanism that prevents over-posting data to the model.

Prevents request forgery

[Bind] attribute – a security mechanism that prevents over-posting data to the model.
Processing the POST request

HTTP POST method.

Validates the forgery token.

Checks if sent data are valid – server side validation, compared to client-side validation (javascript)

Redirects after successful update.

In case of invalid data, the original form is returned back to the client, displaying error messages
HTTP methods – best practices

• HttpGet and HttpPost method overloads
• All methods that modify data SHOULD use HttpPost method overload
• Modifying data in HttpGet method
  – security risk
  – Violates HTTP best practices
  – Violates REST architectural pattern
• GET method SHOULD NOT have any side effect and SHOULD NOT modify persistent data
ADDING SEARCH
Search form – Index.cshtml

Enter a text filtering value.
View/Controller – changes

View (changes)

```csharp
@model IEnumerable<MvcMovie.Models.Movie>
@
    ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>
    @Html.ActionLink("Create New", "Create")
    @using (Html.BeginForm("SearchIndex", "Movies", FormMethod.Get))
    {
        <p>Title: @Html.TextBox("SearchString")<br />
        <input type="submit" value="Filter" /></p>
    }
    @using (Html.BeginForm("Index", "Movies", FormMethod.Get))
    {
        <p>@Html.ActionLink("Create New", "Create")</p>
        <button>Filter</button>
    }
</p>
```

Controller – changed signature of the method Index.

```csharp
public ActionResult Index(string searchString)
{
    var movies = from m in db.Movies
                  select m;
    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }
    return View(movies);
}
```

LINQ query definition (NOT execution!)

Lambda expression

Default form method = POST!

Use overridden BeginForm method to forceHttpGet method.
Searching movies – URL query
Adding search by Genre

HttpGet method handles the request.
Search by Genre – View

```csharp
@model IEnumerable<MvcMovie.Models.Movie>
{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>
    @Html.ActionLink("Create New", "Create")
    @using (Html.BeginForm("Index", "Movies", FormMethod.Get))
    {
        <p>
            Genre: @Html.DropDownList("movieGenre", "All")
            Title: @Html.TextBox("SearchString")
            <input type="submit" value="Filter" />
        </p>
    }
</p>
</table class="table">
```

- **DropDown list markup.**
- **Parameter “movieGenre” is the key for populating dropdown list from ViewBag.**
- **Preselected value.**
Search by Genre – Controller

```csharp
public ActionResult Index(string movieGenre, string searchString)
{
    var GenreLst = new List<string>();

    var GenreQry = from d in db.Movies
                   orderby d.Genre
                   select d.Genre;

    GenreLst.AddRange(GenreQry.Distinct());

    ViewBag.movieGenre = new SelectList(GenreLst);

    var movies = from m in db.Movies
                 select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    if (!String.IsNullOrEmpty(movieGenre))
    {
        movies = movies.Where(x => x.Genre == movieGenre);
    }

    return View(movies);
}
```

Populating the list of genres in ViewBag.

Key movieGenre is the same as the parameter of the dropdown list.
public ActionResult Details(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCodeBadRequest);
    }
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    return View(movie);
}
Delete method - Controller

HttpGet method. Selects an objects and returns Details page.

```csharp
// GET: /Movies/Delete/5
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    return View(movie);
}
```

RULE:
Never use a HttpGet method to modify the model. Opens security holes, architecturally bad!

HttpPost method. Deletes an object having the given id.

```csharp
// POST: /Movies/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(int id)
{
    Movie movie = db.Movies.Find(id);
    db.Movies.Remove(movie);
    db.SaveChanges();
    return Redirect("Index");
}
```

Asp .net maps a segment of URL to a method. Attribute ActionName is necessary to provide valid URL routing. The same URL maps to different action methods, based on used HTTP method.
Data Validation

• Keep Things DRY (Don’t Repeat Yourself)
• Declarative validation rules in one place (Model class)
  – Regular expressions
  – Range validation
  – Length validation
  – NULL values validation
  – Data formatting
• Validation rules enforced before saving changes to the database!
Validation rules – Model

```csharp
public class Movie
{
    public int ID { get; set; }
    [StringLength(60, MinimumLength = 3)]
    public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "\{0:yyyy-MM-dd\}", ApplyFormatInEditMode = true)]
    public DateTime ReleaseDate { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z]+-\s\s*$")]
    [Required]
    [StringLength(30)]
    public string Genre { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    public decimal Price { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z]+-\s\s*$")]
    [StringLength(5)]
    public string Rating { get; set; }
}
```

```csharp
MovieDBContext db = new MovieDBContext();
Movie movie = new Movie();
movie.Title = "Gone with the Wind";
db.Movies.Add(movie);
db.SaveChanges(); // <= Will throw server side validation exception
```
Data Validation - View

Client-side validation: javascript (jQuery).

Validation rules picked up from the model class annotations.

Validation messages derived from the validation constraints in the model class.
 Validation message derived from the validation constraints specified for the given Property (Title)
Data Validation - Controller

HttpGet method displays initial Create form.

HttpPost method that does create a new object.

Server-side data validation check.
DataType attributes

• Provide only hits for the view engine to format the data
• Date, Time, PhoneNumber, EmailAddress,…
• Automatic provision of type specific features
  e.g. “mailto: …” link for EmailAddress
• Do NOT provide any Validation
  (just presentation hints)
DisplayFormat annotation

- Used to explicitly specify format of the data
- Example: redefining the default date format

```csharp
[DisplayFormat(DataFormatString = "\{0:yyyy-MM-dd}\", ApplyFormatInEditMode = true)]
public DateTime EnrollmentDate { get; set; }
```

```csharp
public class Movie
{
    public int ID { get; set; }
    [Required,StringLength(60, MinimumLength = 3)]
    public string Title { get; set; }
    [Display(Name = "Release Date"),DataType(DataType.Date)]
    public DateTime ReleaseDate { get; set; }
    [Required]
    public string Genre { get; set; }
    [Range(1, 100),DataType(DataType.Currency)]
    public decimal Price { get; set; }
    [Required,StringLength(5)]
    public string Rating { get; set; }
}
```

It is possible to specify validation properties in one line!
LAMBDA EXPRESSIONS
Introduction

• Expressions that use special syntax
• Anonymous functions used as data (variables, fields, parameters, return values)
• The anonymous functions are used to create delegates and expression trees
• Lambda expressions particularly helpful for writing LINQ queries
• Available from .NET 4.5
Operator =>

- Interpreted as “goes to”
- Used for declaring a lambda expression
- The same priority as assignment (=)
- Right associative operator
- Separates the parameters and function body

<table>
<thead>
<tr>
<th>Left side</th>
<th>=&gt;</th>
<th>Right side</th>
</tr>
</thead>
<tbody>
<tr>
<td>An Empty parameter list</td>
<td></td>
<td>An expression</td>
</tr>
<tr>
<td>A formal parameter list</td>
<td></td>
<td>A Statement list inside curly brackets</td>
</tr>
<tr>
<td>An implicit parameter list</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Anonymous functions

• Inline statements or expressions
• Used wherever a delegate type is expected
• It can initialize a named delegate
• It can be passed as the parameter where a named delegate type is expected
• Two kinds of anonymous functions
  – Anonymous methods
  – Lambda expressions
Evolution of delegates in C#

delegate void TestDelegate(string s);
static void M(string s)
{
    Console.WriteLine(s);
}

static void Main(string[] args)
{
    // Original delegate syntax required
    // initialization with a named method.
    TestDelegate testDelA = new TestDelegate(M);

    // C# 2.0: A delegate can be initialized with
    // inline code, called an "anonymous method." This
    // method takes a string as an input parameter.
    TestDelegate testDelB = delegate (string s) { Console.WriteLine(s); };

    // C# 3.0. A delegate can be initialized with
    // a lambda expression. The lambda also takes a string
    // as an input parameter (x). The type of x is inferred by the compiler.
    TestDelegate testDelC = (x) => { Console.WriteLine(x); };

    // Invoke the delegates.
    testDelA("Hello. My name is M and I write lines.");
    testDelB("That's nothing. I'm anonymous and ");
    testDelC("I'm a famous author.");

    // Keep console window open in debug mode.
    Console.WriteLine("Press any key to exit.");
    Console.ReadKey();
}
Anonymous method

• No name, no overloading
• Created using the `delegate` keyword
• It is possible to add multiple statements inside its body

```csharp
// Create a handler for a click event.
button1.Click += delegate(System.Object o, System.EventArgs e)
    { System.Windows.Forms.MessageBox.Show("Click!"); };

// Create a delegate.
del void Del(int x);

// Instantiate the delegate using an anonymous method.
Del d = delegate(int k) { /* ... */ };

void StartThread()
{
    System.Threading.Thread t1 = new System.Threading.Thread
       (delegate()
           { System.Console.Write("Hello,");
             System.Console.WriteLine("World!");
           });
    t1.Start();
}
Anonymous method (cont.)

- Scope of the parameters is the anonymous method block
- No jump from inside an anonymous method block to the outside, and vice versa.
- Cannot access `ref` and `out` parameters of an outer scope
- No unsafe code access inside its block
- Not allowed on the left side of the operator `is`
Expression lambdas

• Lambda expression with an expression on the right side of the operator =>
• Used dominantly in construction of expression trees
• (input parameters) => expression
• Parentheses optional if lambda has one param.
• Input parameters separated by comma
Expression lambdas - examples

• \((x, y) \Rightarrow x == y\)
  The parameters types inferred by the compiler

• \((\text{int } x, \text{ string } s) \Rightarrow s.Length > x\)
  Specify types of the parameters when the compiler cannot infer them from the code.

• () \Rightarrow \text{SomeMethod}()
  Zero input parameters specified with empty parentheses.
  Note: a method call cannot be evaluated outside the .NET Framework (e.g. SQL Server)
Statement lambdas

• `(input parameters) => {statement;}`
• Statements enclosed in braces
• The body of a statement lambda can contain multiple statements (in practices, two-three)
• Cannot be used to create expression trees

```csharp
delegate void TestDelegate(string s);
...
TestDelegate myDel = n => { string s = n + " " + "World"; Console.WriteLine(s); }; myDel("Hello");
```
Generic delegates – Func

- **System.Func<T,TResult>**
  - T – argument type,
  - TResult – return type (last type parameter)
- Useful for encapsulating user-defined expressions that are applied to all elements of a data set

```csharp
public delegate TResult Func<TArg0, TResult>(TArg0 arg0)

Func<int, bool> myFunc = x => x == 5;
bool result = myFunc(4); // returns false of course
```
Func delegate (cont.)

- A lambda expression can be passed where `Expression<Func>` type is required
  - `System.Linq.Queryable`

```csharp
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
int oddNumbers = numbers.Count(n => n % 2 == 1);

Output: 5, 1, 3, 9, 7

var firstNumbersLessThan6 = numbers.TakeWhile(n => n < 6);

Output: 5, 4, 1, 3

var firstSmallNumbers = numbers.TakeWhile((n, index) => n >= index);

Output: 5, 4
```

Compiler can infer the type of the parameter `n`. 
Type inference in lambdas

• Compiler can infer the type of the parameters based on:
  – Lambda’s body
  – Parameter’s delegate type

• Example:
  ```csharp
  IEnumerable<Customer> customers = ...
  customers.Where(c => c.City == "London");
  ```
  Standard query operator.
Lambda expressions – general rules

- The lambda must contain the same number of parameters as the delegate type
- Each input parameter in the lambda must be implicitly convertible to its corresponding delegate parameter
- The return value of the lambda (if any) must be implicitly convertible to the delegate’s return type
Lambda expressions - examples

- Func<int,int> f1 = x => x+1;
- Func<int,int> f2 = x => {return x+1;}
- Func<int,int> f3 = (int x) => x +1;
- Func<int,int> f4 = (int x) => {return x+1;}
- Func<int,int> f7 = delegate(int x) {return x+1;}

Invocation example:
Console.WriteLine(f1.Invoke(4));
Lambda expressions - examples

• **Func<int,int,int> f5= (x,y) => x*y**
  Invocation: Console.WriteLine(f5.Invoke(2,2));

• **Action f6 = () => Console.WriteLine();**
  Function instance that does not receive any parameter nor returns value.
  Invocation: f6.Invoke();

• **Func<int> f8 = delegate { return 1+1;};**
  Invocation: Console.WriteLine(f8());
Language Integrated Query

LINQ
Content

• Understand what LINQ is?
• Learn what problems solves
• See what its syntax looks like
• Know where LINQ can be used
What is LINQ?

• Language INtegrated Query
• It is part of programming language syntax
• Supported by: C#, VB, Delphi Prism
• Used for querying data
• Supported the following types of data sources
  – Relational data
  – XML data
  – objects
LINQ Architecture
ADO .NET vs. LINQ

ADO .NET
• OO library for relational data access
• Mapping from relational to OO objects needed!
• High Impedance Missmatch for mapping data from storage to objects in an application

LINQ
• SQL-Like syntax that deals with pure objects
• Reduces the Impedance Missmatch
• Makes data querying more efficient
• One still must know the format of the data
LINQ Adapters

- LINQ to Objects
- LINQ to SQL
- LINQ to XML
- LINQ to Entities
- It is possible to create own customized adapter
  - E.g. LINQ for Querying Twitter API
References

• ASP .NET MVC 5 Tutorial – Official
  http://www.asp.net/mvc/tutorials/mvc-5/introduction/getting-started

• Lambda expressions
  http://www.dotnetperls.com/lambda

• LINQ
  http://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b