Articles » Web Development » ASP.NET » General

Learn MVC Project in 7 days - Day 6

Marla Sukesh, 27 Jul 2015

This article is continuation of day 6 for Learn MVC Project in 7 Days.

Download Day_6_file_upload.zip
Download Day_6_exception.zip
Download Day_6_Routing.zip

Introduction

Welcome to Day 6 of "Learn MVC Project in 7 days" series. Hope you had a nice time reading Day 1 to Day 5. Completing previous days is must prior to Day 6.

Complete Series

- Day 1
- Day 2
- Day 3
- Day 4
- Day 5
- Day 6
- Day 7
- Bonus Day 1
- Bonus Day 2

We are pleased to announce that this article is now available as hard copy book you can get the same from www.amazon.com and www.flipkart.com

Agenda

Lab 27 – Add Bulk upload option

In this lab we will create an option for uploading multiple employees from a CSV file.

We will learn two new things here.

1. How to use File upload control
2. Asynchronous controllers.

Step 1 – Create FileUploadViewModel

Create a new class called FileUploadViewModel in ViewModels folder as follows.

```
public class FileUploadViewModel: BaseViewModel
{
    public HttpPostedFileBase fileUpload {get; set ;}
}
```

HttpPostedFileBase will provide the access to the uploadedfile by client.

Step 2 - Create BulkUploadController and Index Action

Create a new controller called BulkUploadController and an action method called Index Action as follows.

```
public class BulkUploadController : Controller
{
        [HeaderFooterFilter]
        [AdminFilter]
        public ActionResult Index()
        {
            return View(new FileUploadViewModel());
        }
}
```

As you can see, Index action is attached with HeaderFooterFilter and AdminFilter attributes. HeaderFooterFilter makes sure that correct header and footer data is passed to the ViewModel and AdminFilter restricts access to action method by Non-Admin user.

Step 3 – Create Upload View

Create a view for above action method.

Kindly note that view name should be index.cshtml and should be placed in "~/Views/BulkUpload" folder.

Step 4 – Design Upload View

Put following contents inside the view.

```
@using WebApplication1.ViewModels
@model FileUploadViewModel
@{
    Layout = "~/Views/Shared/MyLayout.cshtml";
}

@section TitleSection{
    Bulk Upload
}
@section ContentBody{
    <div>
    <a href="/Employee/Index">Back</a>
        <form action="/BulkUpload/Upload" method="post" enctype="multipart/form-data">
            Select File : <input type="file" name="fileUpload" value="" />
            <input type="submit" name="name" value="Upload" />
        </form>
    </div>
}
```

As you can see name of the property in FileUploadViewModel and name of the input [type="file"] are same. It is “fileUpload”. We spoke on the importance of name attribute in Model Binder lab. Note: In form tag one additional attribute that is enctype is specified. We will talk about it in the end of the lab.

Step 5 - Create Business Layer Upload method

Create a new method calledUploadEmployees inEmployeeBusinessLayer as follows.

```
public void UploadEmployees(List<Employee> employees)
{
    SalesERPDAL salesDal = new SalesERPDAL();
    salesDal.Employees.AddRange(employees);
    salesDal.SaveChanges();
}<employee>
</employee>
```

Step 6 – Create Upload Action method

Create a new action method called Upload inside BulkUploadController as follows.

```
[AdminFilter]
public ActionResult Upload(FileUploadViewModel model)
{
    List<Employee> employees = GetEmployees(model);
    EmployeeBusinessLayer bal = new EmployeeBusinessLayer();
    bal.UploadEmployees(employees);
    return RedirectToAction("Index","Employee");
}

private List<Employee> GetEmployees(FileUploadViewModel model)
{
    List<Employee> employees = new List<Employee>();
    StreamReader csvreader = new StreamReader(model.fileUpload.InputStream);
    csvreader.ReadLine(); // Assuming first line is header
    while (!csvreader.EndOfStream)
    {
        var line = csvreader.ReadLine();
        var values = line.Split(',');//Values are comma separated
        Employee e = new Employee();
        e.FirstName = values[0];
        e.LastName = values[1];
        e.Salary = int.Parse(values[2]);
        employees.Add(e);
    }
    return employees;
}
```

AdminFilter attached to the Upload Action restrict access to Non-Admin user.

Step 7 – Create a link for BulkUpload

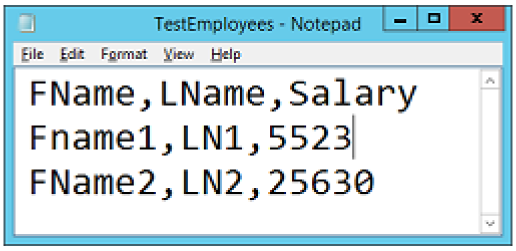
Open AddNewLink.cshtml from “Views/Employee” folder and put a link for BulkUpload as follows.

```
<a href="/Employee/AddNew">Add New</a>
&nbsp;
&nbsp;
<a href="/BulkUpload/Index">BulkUpload</a>
```

Step 8 – Execute and Test

Step 8.1 – Create a sample file for testing

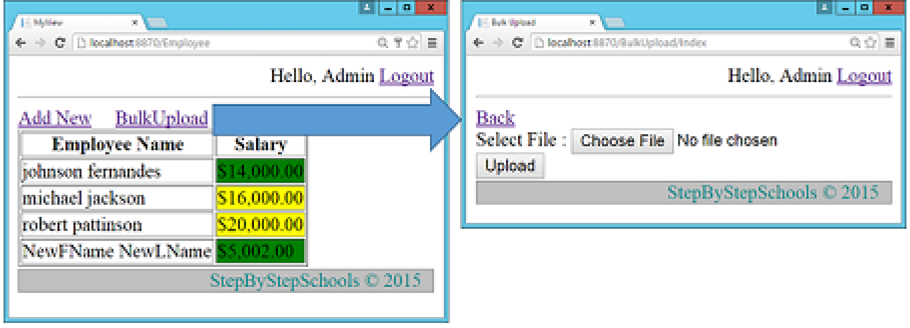
Create a sample file like below and save it somewhere in the computer.



```
FName,LName,Salary
Fname1,LN1,5523
FName2,LN2,25630
```

Step 8.2 – Execute and Test

Press F5 and execute the application. Complete the login process and navigate to BulkUpload option by clicking link.


localhost:8870/Employee

Hello, Admin Logout

Add New BulkUpload

Employee Name	Salary
johnson fernandes	$14,000.00
michael jackson	$16,000.00
robert pattinson	$20,000.00
NewFName NewLName	$5,002.00

StepByStepSchools © 2015

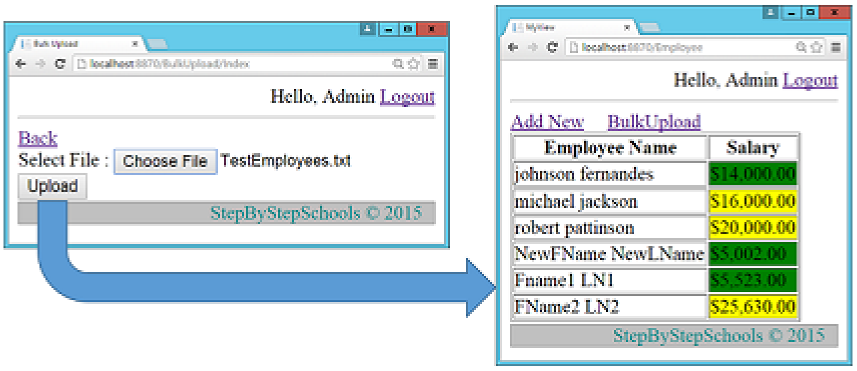
localhost:8870/BulkUpload/Index

Hello, Admin Logout

Back
Select File : Choose File No file chosen
Upload

StepByStepSchools © 2015

Select File and Click on Upload.


localhost:8870/BulkUpload/Index

Hello, Admin Logout

Back
Select File : Choose File TestEmployees.txt
Upload

StepByStepSchools © 2015

localhost:8870/Employee

Hello, Admin Logout

Add New BulkUpload

Employee Name	Salary
johnson fernandes	$14,000.00
michael jackson	$16,000.00
robert pattinson	$20,000.00
NewFName NewLName	$5,002.00
Fname1 LN1	$5,523.00
FName2 LN2	$25,630.00

StepByStepSchools © 2015

Note:

In above example we have not applied any client side or server side validation in the View. It may leads to following error.
“Validation failed for one or more entities. See 'EntityValidationErrors' property for more details.”

To find the exact cause for the error, simply add a watch with following watch expression when exception occurs.
((System.Data.Entity.Validation.DbEntityValidationException)$exception).EntityValidationErrors

The watch expression ‘$exception’ displays any exception thrown in the current context, even if it has not been caught and assigned to a variable.

Talk on Lab 27

Why don’t we have validations here?

Adding client side and server side validation to this option will be an assignment for readers. I will give you a hint.

- For Server side validation use Data Annotations.
- For client side either you can leverage data annotation and implement jQuery unobtrusive validation. Obviously this time you have to set custom data attributes manually because we don’t have readymade Htmlhelper method for file input.
  Note: If you didn’t understood this point, I recommend you to go through “implanting client side validation in Login view” again.
- For client side validation you can write custom JavaScript and invoke it on button click. This won’t be much difficult because file input is an input control at the end of the day and its value can be retrieved inside JavaScript and can be validated.

What is HttpPostedFileBase?

HttpPostedFileBase will provide the access to the file uploaded by client. Model binderwill update the value of all properties FileUploadViewModel class during post request. Right now we have only one property inside FileUploadViewModel and Model Binder will set it to file uploaded by client.

Is it possible to provide multiple file input control?

Yes, we can achieve it in two ways.

1. Create multiple file input controls. Each control must have unique name. Now in FileUploadViewModel class create a property of type HttpPostedFileBase one for each control. Each property name should match with the name of one control. Remaining magic will be done by ModelBinder. ☺
2. Create multiple file input controls. Each control must have same name. Now instead of creating multiple properties of type HttpPostedFileBase, createone of type List<httppostedfilebase>.

Note: Above case is true for all controls. When you have multiple controls with same name ModelBinder update the property with the value of first control if property is simple parameter. ModelBinder will put values of each control in a list if property is a list property.

What does enctype="multipart/form-data" will do?

Well this is not a very important thing to know but definitely good to know.

This attribute specifies the encoding type to be used while posting data.

The default value for this attribute is "application/x-www-form-urlencoded"

Example – Our login form will send following post request to the server

```
POST /Authentication/DoLogin HTTP/1.1
Host: localhost:8870
Connection: keep-alive
Content-Length: 44
Content-Type: application/x-www-form-urlencoded
...
...
UserName=Admin&Passsword=Admin&BtnSubmi=Login
```

All input values are sent as one part in the form of key/value pair connected via "&".

Whenenctype="multipart/form-data" attribute is added to form tag, following post request will be sent to the server.

```
POST /Authentication/DoLogin HTTP/1.1
Host: localhost:8870
Connection: keep-alive
Content-Length: 452
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarywHxplIF8cR8KNjeJ
...
...
------WebKitFormBoundary7hciuLuSNglCR8WC
Content-Disposition: form-data; name="UserName"

Admin
------WebKitFormBoundary7hciuLuSNglCR8WC
Content-Disposition: form-data; name="Password"

Admin
------WebKitFormBoundary7hciuLuSNglCR8WC
Content-Disposition: form-data; name="BtnSubmi"

Login
------WebKitFormBoundary7hciuLuSNglCR8WC--
```

As you can see, form is posted in multiple part. Each part is separated by a boundary defined by Content-Type and each part contain one value.

encType must be set to "multipart/form-data" if form tag contains file input control.

Note: boundary will be generated randomly every time request is made. You may see some different boundary.

Why don't we always set encType to "multipart/form-data"?

When encType is set to "multipart/form-data", it will do both the things–Post the data and upload the file. Then why don't we always set it as "multipart/form-data".

Answer is, it will also increase the overall size of the request. More size of the request means less performance. Hence as a best practice we should set it to default that is "application/x-www-form-urlencoded".

Why we have created ViewModel here?

We had only one control in our View. We can achieve same result by directly adding a parameter of type HttpPostedFileBase with name fileUpload in Upload action method Instead of creating a separate ViewModel. Look at the following code.

```
public ActionResult Upload(HttpPostedFileBase fileUpload)
{
}
```

Then why we have created a separate class.

Creating ViewModel is a best practice. Controller should always send data to the view in the form of ViewModel and data sent from view should come to controller as ViewModel.

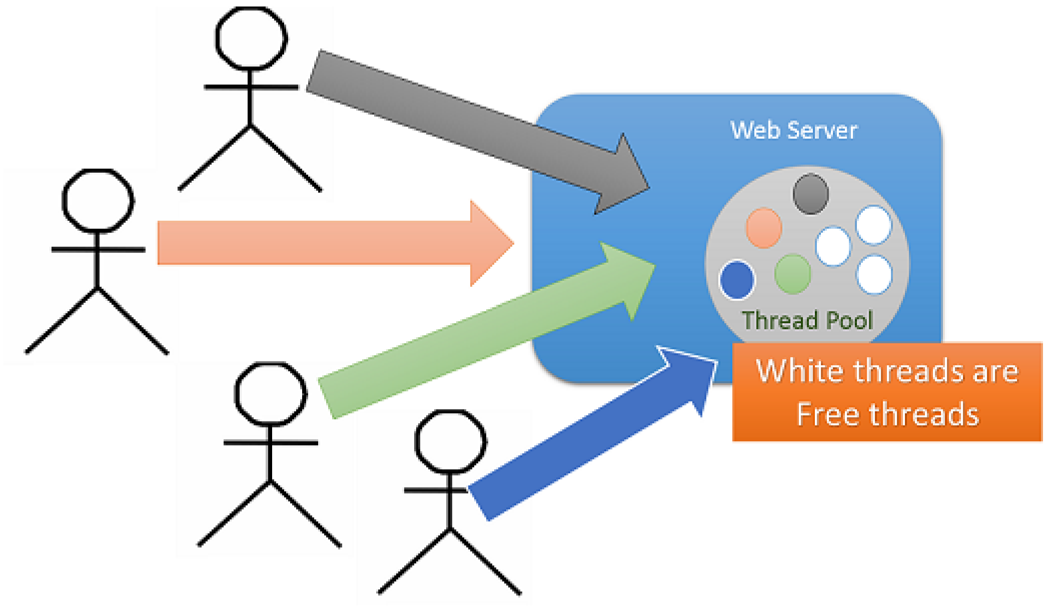
Problem in the above solution

Did you ever wondered how you get response when you send a request?
Now don't say, action method receive request and blah blah blah!!! ☺

Although it's the correct answer I was expecting a little different answer.
My question is what happen in the beginning.

A simple programming rule – everything in a program is executed by a thread even a request.

In case of Asp.net on the webserver .net framework maintains a pool of threads.
Each time a request is sent to the webserver a free thread from the pool is allocated to serve the request. This thread will be called as worker thread.



Worker thread will be blocked while the request is being processed and cannot serve another request.

Now let's say an application receives too many requests and each request will take long time to get completely processed. In this case we may end up at a point where new request will get into a state where there will be noworker thread available to serve that request. This is called as Thread Starvation.

In our case sample file had 2 employee records but in real time it may contain thousands or may be lacks of records. It means request will take huge amount of time to complete the processing. It may leads to Thread Starvation.

Solution

Now the request which we had discussed so far is of type synchronous request.

Instead of synchronous if client makesan asynchronous request, problem of thread starvation get solved.

- In case of asynchronous request as usual worker thread from thread pool get allocated to serve the request.
- Worker thread initiates the asynchronous operation and returned to thread pool to serve another request. Asynchronous operation now will be continued by CLR thread.
- Now the problem is, CLR thread can't return response so once it completes the asynchronous operation it notifies ASP.NET.
- Webserver again gets a worker thread from thread pool and processes the remaining request and renders the response.

In this entire scenario two times worker thread is retrieved from thread pool. Now both of them may be same thread or they may not be.

Now in our example file reading is an I/O bound operation which is not required to be processed by worker thread. So it's a best place to convert synchronous requests to asynchronous requests.

Does asynchronous request improves response time?

No, response time will remain same. Here thread will get freed for serving other requests.

Lab 28 – Solve thread starvation problem

In ASP.NET MVC we can convert synchronous requests to asynchronous requests by converting synchronous action methods to asynchronous action methods.

Step 1 - Create asynchronous Controller

Change base class of UploadController to AsynController from Controller.

```
{
    public class BulkUploadController : AsyncController
    {
```

Step 2 – Convert synchronous action method to asynchronous action method

It can be done easily with the help of two keywords – async and await.

```
[AdminFilter]
public async Task<ActionResult> Upload(FileUploadViewModel model)
{
    int t1 = Thread.CurrentThread.ManagedThreadId;
    List<Employee> employees = await Task.Factory.StartNew<List<Employee>>
        (() => GetEmployees(model));
    int t2 = Thread.CurrentThread.ManagedThreadId;
    EmployeeBusinessLayer bal = new EmployeeBusinessLayer();
    bal.UploadEmployees(employees);
    return RedirectToAction("Index", "Employee");
}<actionresult><employee><list<employee>
</list<employee></employee></actionresult>
```

As you can see we are storing thread id in a variable in the beginning and in the end of action method.

Let's understand the code.

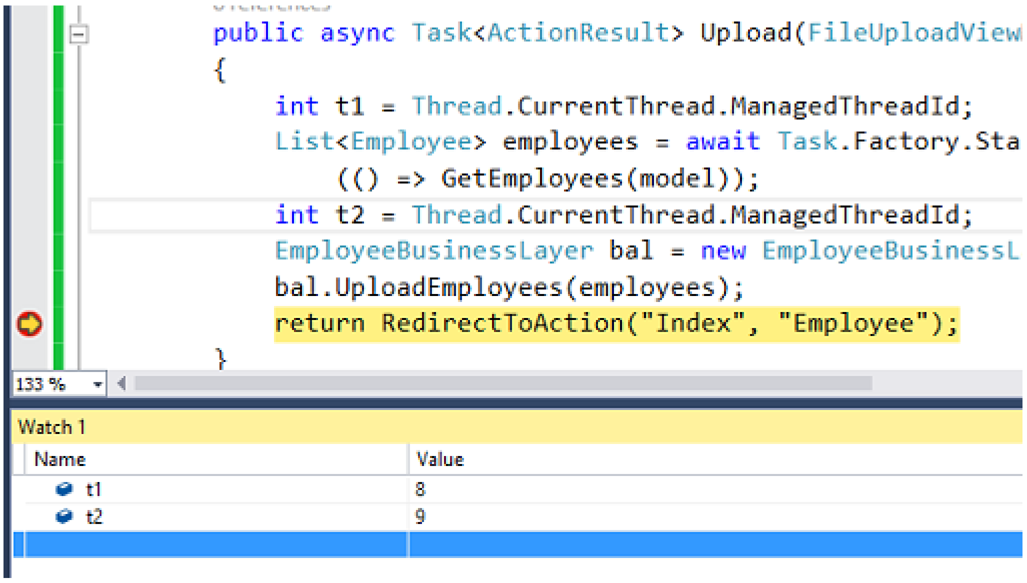
- When upload button is clicked by the client, new request will be sent to the server.
- Webserver will take a worker thread from thread pool and allocate it to serve the request.
- Worker thread make action method to execute.
- Worker method starts an asynchronous operation with the help of Task.Factory.StartNew method.
- As you can see action method is marked as asynchronous with the help of async keyword. It will make sure that worker thread get released as soon as asynchronous operation starts.Now logically asynchronous operation will continue its execution in the background by a separate CLR thread.
- Now asynchronous operation call is marked with await keyword. It will make sure that next line wont executes unless asynchronous operation completes.
- Once Asynchronous operation completes next statement in the action method should execute and for that again a worker thread is required. Hence webserver will simply take up a new free worker thread from thread pool and allocate it to serve the remaining request and to render response.

Step 3 – Execute and Test

Execute the application. Navigate to BulkUpload option.

Now before you do anything more in the output, navigate to code and put a breakpoint at the last line.

Now select the sample file and click on Upload.



```
public async Task<ActionResult> Upload(FileUploadView
{
    int t1 = Thread.CurrentThread.ManagedThreadId;
    List<Employee> employees = await Task.Factory.Sta
        (() => GetEmployees(model));
    int t2 = Thread.CurrentThread.ManagedThreadId;
    EmployeeBusinessLayer bal = new EmployeeBusinessL
    bal.UploadEmployees(employees);
    return RedirectToAction("Index", "Employee");
}
```

Name	Value
t1	8
t2	9

As you can see we have different thread id in the beginning and different in end. Output is going to be same as previous lab.

Lab 29 – Exception Handling – Display Custom error page

A project will not be considered as a complete project without a proper exception handling.

So far we have spoken about two filters in Asp.Net MVC – Action Filters and Authorization Filters. Now it's time for third one – Exception Filters.

What are Exception Filters?

Exception Filters will be used in the same way other filters are used. We will use them as an attribute.

Steps for using exception filter

- Enable them.
- Apply them as an attribute to an action method or to a controller. We can also apply exception filter at global level.

<u>What they will do?</u>

Exception filter will take control of the execution and start executing code written inside it automatically as soon exception occurs inside action method.

<u>Is there any automation?</u>

ASP.NET MVC provides us one readymade Exception Filter called HandeError.

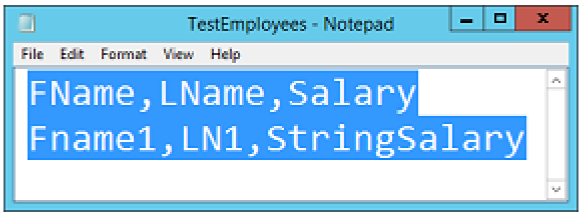
As we said before it will execute when exception occurs in an action method. This filter will find a view inside "~/Views/[current controller]" or "~/Views/Shared" folder with name "Error" , create the ViewResult of that view and return as a response.

Let's see a demo and understand it better.

In the last lab in our project we have implemented BulkUpload option. Now there is high possibility of error in the input file.

Step 1 – Create a sample Upload file with Error

Create a sample upload file just like before but this time put some invalid value.

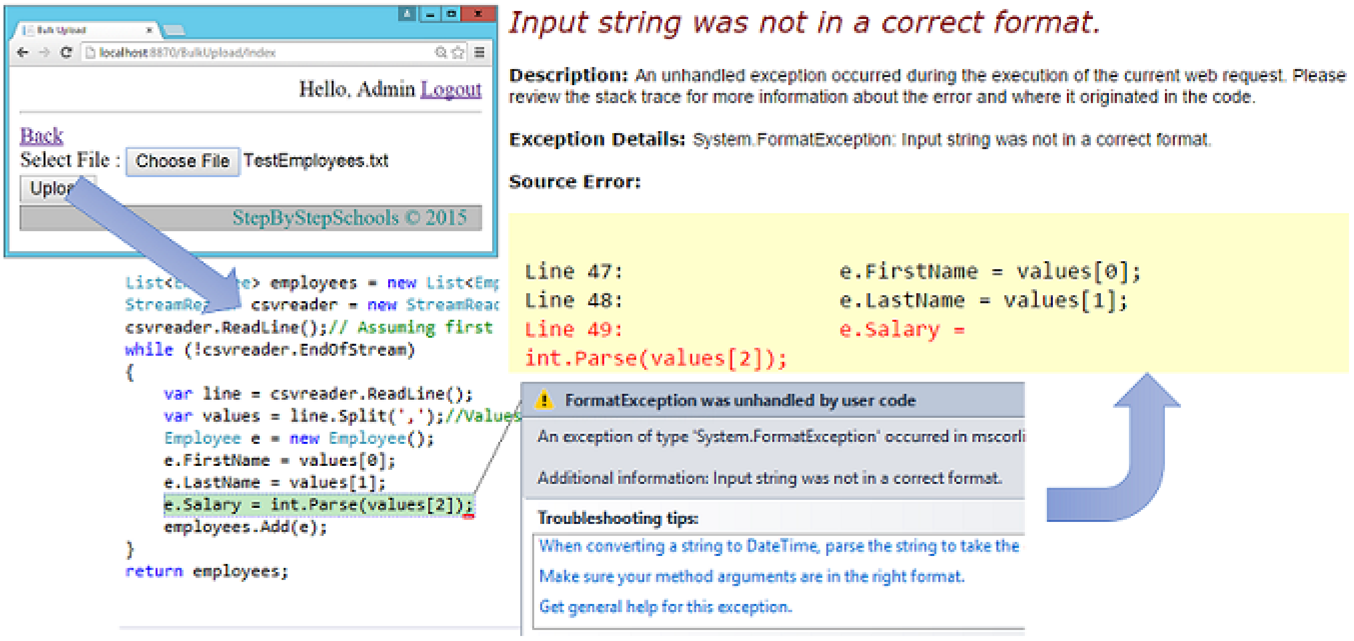


```
FName,LName,Salary
Fname1,LN1,StringSalary
```

As you can see, Salary is invalid.

Step 2 – Execute and Test for exception

Press F5 and execute the application. Navigate to Bulk Upload Option. Select above file and click on Upload.


Bulk Upload
localhost:8870/BulkUpload/Index
Hello, Admin Logout
Back
Select File : Choose File TestEmployees.txt
Uplo
StepByStepSchools © 2015

```
List<E      e> employees = new List<Emp
StreamRe     csvreader = new StreamReac
csvreader.ReadLine();// Assuming first
while (!csvreader.EndOfStream)
{
    var line = csvreader.ReadLine();
    var values = line.Split(',');//Values
    Employee e = new Employee();
    e.FirstName = values[0];
    e.LastName = values[1];
    e.Salary = int.Parse(values[2]);
    employees.Add(e);
}
return employees;
```

Input string was not in a correct format.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.FormatException: Input string was not in a correct format.

Source Error:

```
Line 47:                e.FirstName = values[0];
Line 48:                e.LastName = values[1];
Line 49:                e.Salary =
int.Parse(values[2]);
```

FormatException was unhandled by user code

An exception of type 'System.FormatException' occurred in mscorli

Additional information: Input string was not in a correct format.

Troubleshooting tips:

When converting a string to DateTime, parse the string to take the
Make sure your method arguments are in the right format.
Get general help for this exception.

Step 3 – Enable Exception Filters

Exception filters get enabled when custom exception is enabled. To enable Custom Exception, open web.config file and navigate to System.Web Section. Add a new section for custom errors as below.

```
<system.web>
   <customErrors mode="On"></customErrors>
```

Step 4 – Create Error View

In "~/Views/Shared" folder you will find a view called "Error.cshtml". This file was created as a part ofMVC template in the beginning only. In case if it is not created ,create it manually.

```
@{
    Layout = null;
}
```

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Error</title>
</head>
<body>
    <hgroup>
        <h1>Error.</h1>
        <h2>An error occurred while processing your request.</h2>
    </hgroup>
</body>
</html>
```

<hgroup>

Step 5 – Attach Exception filter

As we discussed before after we enable exception filter we attach them to action method or to controller.

Good news.☺ It's not required to manually attach it.

Open FilterConfig.cs file from App_Start folder. In RegisterGlobalFilters method you will see that HandleError filter is already attached at global level.

```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    filters.Add(new HandleErrorAttribute());//ExceptionFilter
    filters.Add(new AuthorizeAttribute());
}
```

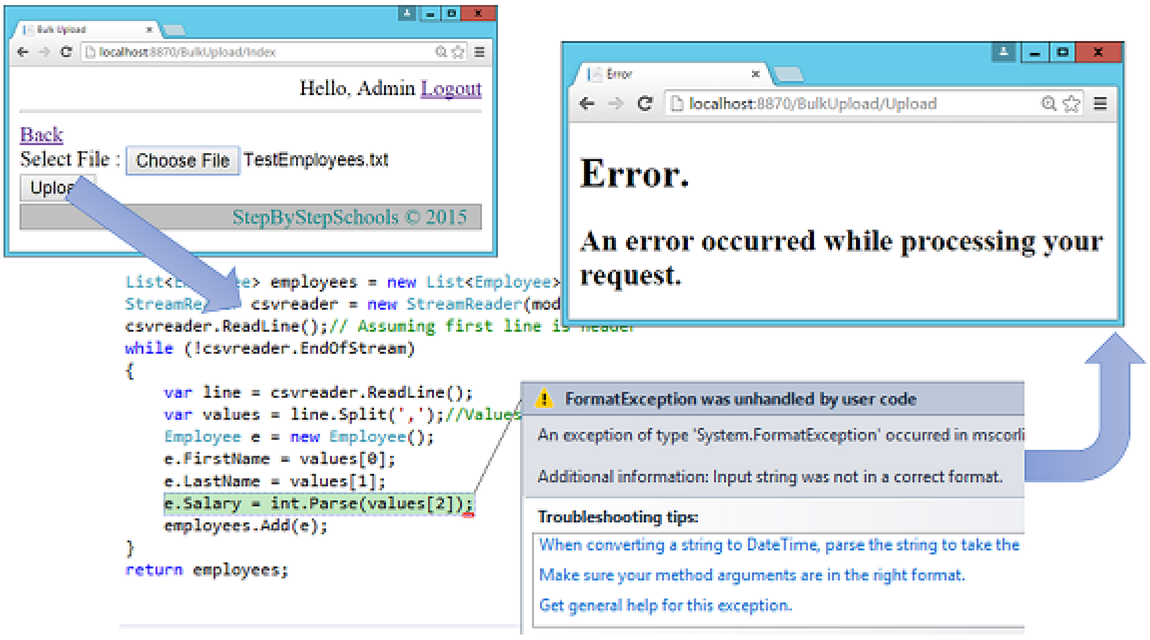
If required remove global filter and attach it at action or controller level as below.

```
[AdminFilter]
[HandleError]
public async Task<ActionResult> Upload(FileUploadViewModel model)
{<actionresult>
</actionresult>
```

It's not recommended to do it. It's better to apply at global level.

Step 6 – Execute and Test

Let's test the application in the same way we did before.


Bulk Upload
localhost:8870/BulkUpload/Index
Hello, Admin Logout
Back
Select File : Choose File TestEmployees.txt
Uplo
StepByStepSchools © 2015
Error
localhost:8870/BulkUpload/Upload

Error.

An error occurred while processing your request.

```
List<E   ee> employees = new List<Employee>
StreamRe    csvreader = new StreamReader(mod
csvreader.ReadLine();// Assuming first line is
while (!csvreader.EndOfStream)
{
    var line = csvreader.ReadLine();
    var values = line.Split(',');//Values
    Employee e = new Employee();
    e.FirstName = values[0];
    e.LastName = values[1];
    e.Salary = int.Parse(values[2]);
    employees.Add(e);
}
return employees;
```

FormatException was unhandled by user code

An exception of type 'System.FormatException' occurred in mscorli

Additional information: Input string was not in a correct format.

Troubleshooting tips:

When converting a string to DateTime, parse the string to take the

Make sure your method arguments are in the right format.

Get general help for this exception.

Step 7 – Display Error message in the view

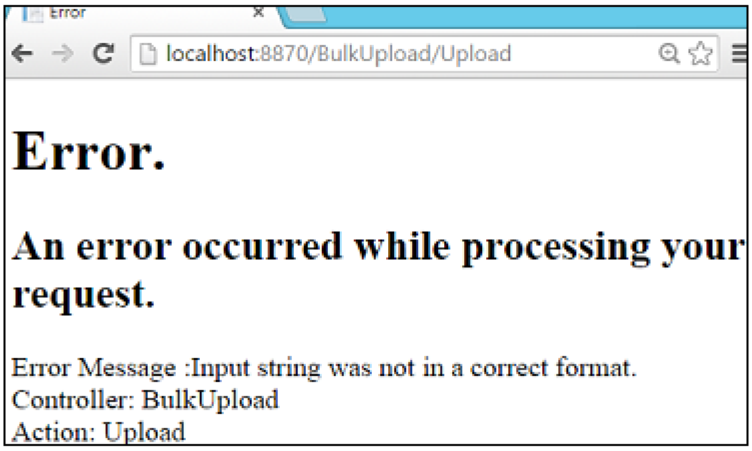
In order to achieve this convert Error view to strongly typed view of type HandleErrorInfo class and then display error messages in the view.

```
@model HandleErrorInfo
@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Error</title>
</head>
<body>
    <hgroup>
        <h1>Error.</h1>
        <h2>An error occurred while processing your request.</h2>
    </hgroup>
        Error Message :@Model.Exception.Message<br />
        Controller: @Model.ControllerName<br />
        Action: @Model.ActionName
</body>
</html>
```

Step 8 – Execute and Test

Perform the same testing but this time we will get following Error view.



Error.

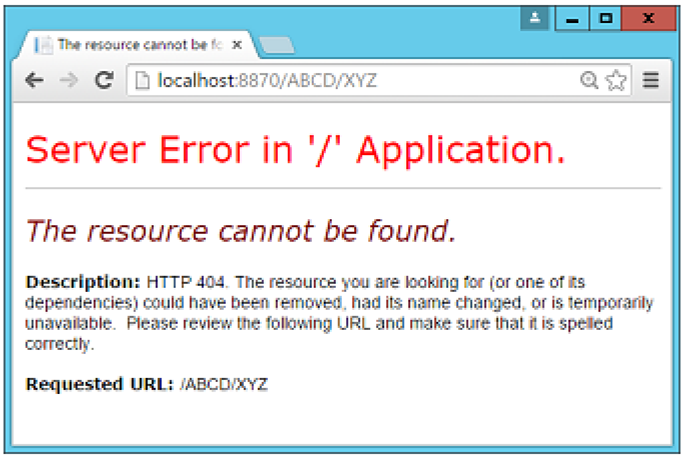
An error occurred while processing your request.

Error Message :Input string was not in a correct format.
Controller: BulkUpload
Action: Upload

Are we missing something?

Handle error attribute make sure that custom view get displayed whenever exception occurs in an action method. But Its power is limited to controller and action methods. It will not handle “Resource not found” error.

Execute the application and put something weird in URL



Server Error in '/' Application.

The resource cannot be found.

Description: HTTP 404. The resource you are looking for (or one of its dependencies) could have been removed, had its name changed, or is temporarily unavailable. Please review the following URL and make sure that it is spelled correctly.

Requested URL: /ABCD/XYZ

Step 9 – Create ErrorController as follows

Create a new controller called ErrorController in Controller folder and create an action method called Index as follows.

```
public class ErrorController : Controller
{
    // GET: Error
    public ActionResult Index()
    {
        Exception e=new Exception("Invalid Controller or/and Action Name");
        HandleErrorInfo eInfo = new HandleErrorInfo(e, "Unknown", "Unknown");
        return View("Error", eInfo);
    }
}
```

HandleErrorInfo constructor takes 3 arguments – Exception object, controller nameand Action method Name.

Step 10 – Display Custom Error View on Invalid URL

In web.config define setting for “Resource not found error” as follows.

```
<system.web>
    <customErrors mode="On">
      <error statusCode="404" redirect="~/Error/Index"/>
    </customErrors>
```

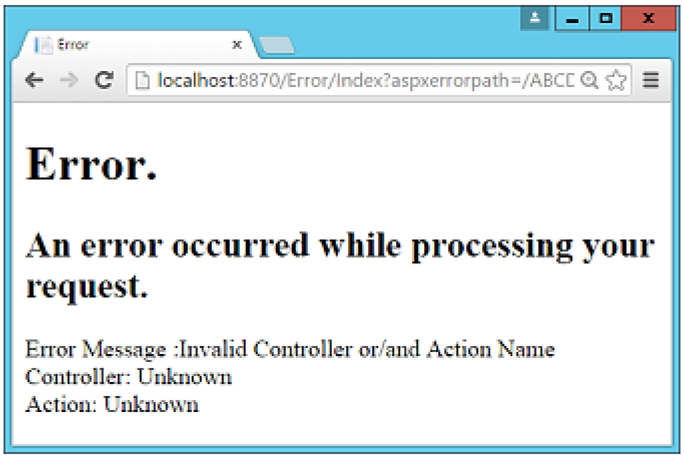
Step 11 - Make ErrorController accessible to everyone

Apply AllowAnonymous attribute to ErrorController because error controller and index action should not be bound to an authenticated user. It may be possible that user has entered invalid URL before login.

```
[AllowAnonymous]
public class ErrorController : Controller
{
```

Step 12 - Execute and Test

Execute the application and put some invalid URL in the address bar.



Error.

An error occurred while processing your request.

Error Message :Invalid Controller or/and Action Name
Controller: Unknown
Action: Unknown

Talk on Lab 29

Is it possible to change the view name?

Yes, it’s not required to keep view name as “Error” always.

In that case we have to specify view name while attaching HandlError filter.

```
[HandleError(View="MyError")]
Or
filters.Add(new HandleErrorAttribute()
                {
                    View="MyError"
                });
```

Is it possible to get different error view for different exceptions?

Yes, it is possible. In that case we have to apply handle error filter multiple times.

```
[HandleError(View="DivideError",ExceptionType=typeof(DivideByZeroException))]
[HandleError(View = "NotFiniteError", ExceptionType = typeof(NotFiniteNumberException))]
[HandleError]

OR

filters.Add(new HandleErrorAttribute()
    {
        ExceptionType = typeof(DivideByZeroException),
        View = "DivideError"
    });
filters.Add(new HandleErrorAttribute()
{
    ExceptionType = typeof(NotFiniteNumberException),
    View = "NotFiniteError"
});
filters.Add(new HandleErrorAttribute());
```

In above case we are adding Handle Error filter thrice. First two are specific to exception whereas last one is more general one and it will display Error View for all other exceptions.

Understand limitation in above lab

The only limitation with above lab is we are not logging our exception anywhere.

Lab 30 – Exception Handling – Log Exception

Step 1 – Create Logger class

Create a new Folder called Logger in root location of the project.

Create a new class called FileLogger as follows inside Logger folder.

```
namespace WebApplication1.Logger
{
    public class FileLogger
    {
        public void LogException(Exception e)
        {
            File.WriteAllLines("C://Error//" + DateTime.Now.ToString("dd-MM-yyyy mm hh ss")+".txt",
                new string[]
                {
                    "Message:"+e.Message,
                    "Stacktrace:"+e.StackTrace
                });
        }
    }
}
```

Step 2 – Create EmployeeExceptionFilter class

Create a new class called EmployeeExceptionFilter inside Filters folder as follows.

```
namespace WebApplication1.Filters
{
    public class EmployeeExceptionFilter
    {
    }
}
```

Step 3 - Extend Handle Error to implement Logging

Inherit EmployeeExceptionFilter from HandleErrorAttribute class and override OnException method as follows.

```
public class EmployeeExceptionFilter:HandleErrorAttribute
{
    public override void OnException(ExceptionContext filterContext)
    {
        base.OnException(filterContext);
    }
}
```

Note: Make sure to put using System.Web.MVC in the top.HandleErrorAttribute class exists inside this namespace.

Step 4 – Define OnException method

Include Exception logging code inside OnException method as follows.

```
public override void OnException(ExceptionContext filterContext)
{
    FileLogger logger = new FileLogger();
    logger.LogException(filterContext.Exception);
    base.OnException(filterContext);
}
```

Step 5 – Change Default Exception filter

Open FilterConfig.cs file and remove HandErrorAtrribute and attach the one we created in last step as follows.

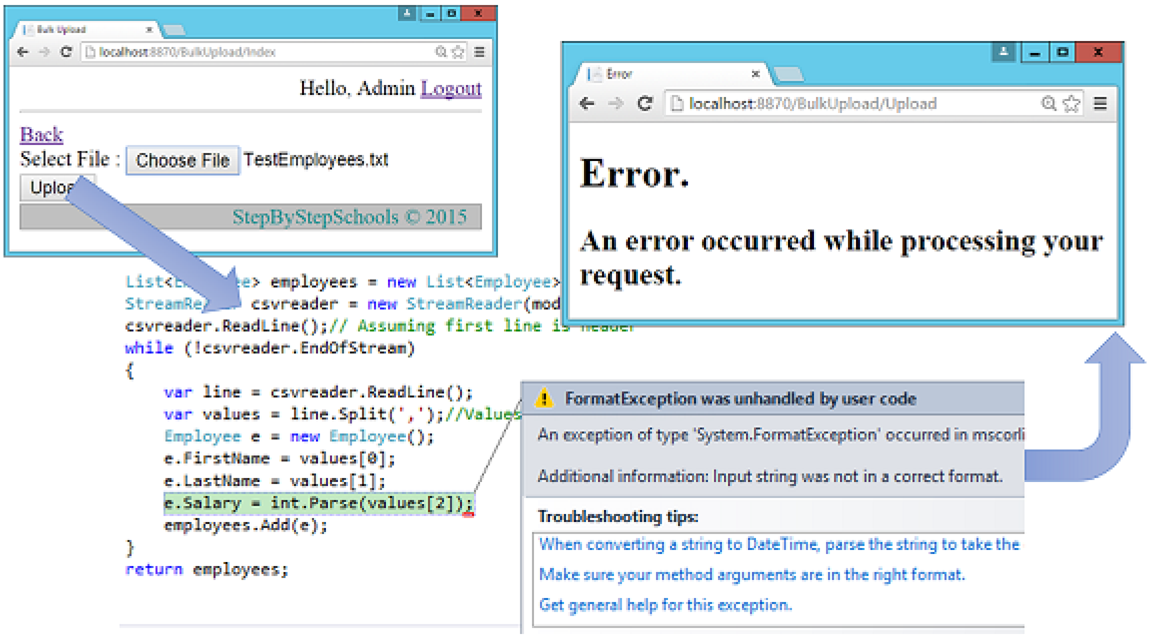
```
public static void RegisterGlobalFilters(GlobalFilterCollection filters)
{
    //filters.Add(new HandleErrorAttribute());//ExceptionFilter
    filters.Add(new EmployeeExceptionFilter());
    filters.Add(new AuthorizeAttribute());
}
```

Step 6 – Execute and Test

First of all create a folder called “Error” in C drive because that’s the place where error files are going to be placed.

Note: If required change the path to your desired one.

Press F5 and execute the application. Navigate to Bulk Upload Option. Select above file and click on Upload.


localhost:8870/BulkUpload/Index
Hello, Admin Logout
Back
Select File : Choose File TestEmployees.txt
Upload
StepByStepSchools © 2015
localhost:8870/BulkUpload/Upload

Error.

An error occurred while processing your request.

```
List<Employee> employees = new List<Employee>
StreamReader csvreader = new StreamReader(mod
csvreader.ReadLine();// Assuming first line is header
while (!csvreader.EndOfStream)
{
    var line = csvreader.ReadLine();
    var values = line.Split(',');//Values
    Employee e = new Employee();
    e.FirstName = values[0];
    e.LastName = values[1];
    e.Salary = int.Parse(values[2]);
    employees.Add(e);
}
return employees;
```

FormatException was unhandled by user code

An exception of type 'System.FormatException' occurred in mscorli

Additional information: Input string was not in a correct format.

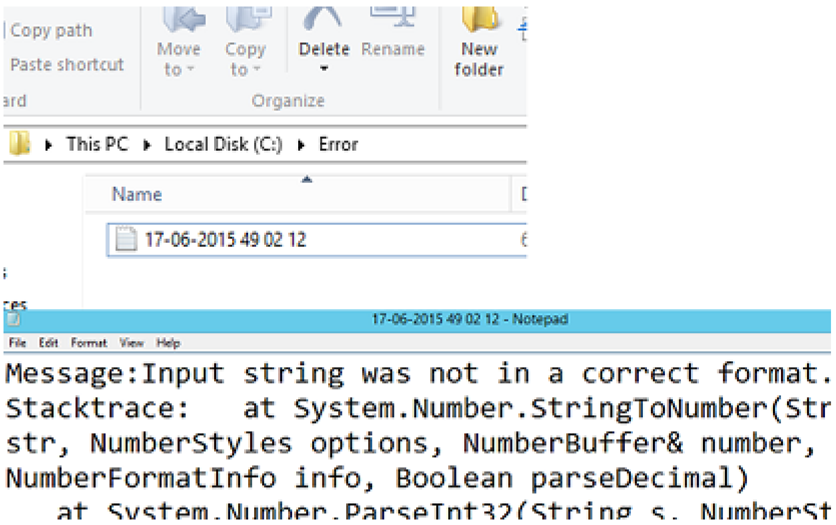
Troubleshooting tips:

When converting a string to DateTime, parse the string to take the

Make sure your method arguments are in the right format.

Get general help for this exception.

Output won’t be different this time. We will get a same Error View like before. Only difference will be this time we will also find an error file created in “C:\\Errors” folder.



```
Message:Input string was not in a correct format.
Stacktrace:   at System.Number.StringToNumber(Str
str, NumberStyles options, NumberBuffer& number,
NumberFormatInfo info, Boolean parseDecimal)
   at System.Number.ParseInt32(String s, NumberSt
```

Talk on Lab 30

How come Error view is returned as a response when exception occurs?

In above lab we have overridden the OnException method and implemented exception logging functionality. Now the question is, how come the default handle error filter is still working then? It’s simple. Check the last line in the OnException method.

```
base.OnException(filterContext);
```

It means, let base class OnException do the reaming work and base class OnException will return ViewResult of the Error View.

Can we return some other result inside OnException?

Yes. Look at the following code.

```
public override void OnException(ExceptionContext filterContext)
{
    FileLogger logger = new FileLogger();
    logger.LogException(filterContext.Exception);
    //base.OnException(filterContext);
    filterContext.ExceptionHandled = true;
    filterContext.Result = new ContentResult()
    {
        Content="Sorry for the Error"
    };
}
```

When we want to return custom response the first thing we should do is, inform MVC engine that we have handled exception manually so don't perform the default behaviour that is don't display the default error screen. This will be done with following statement.

```
filterContext.ExceptionHandled = true
```

You can read more about exception handling in ASP.NET MVC here

Routing

So far we have discussed many concepts, we answered many questions in MVC except one basic and important one.

"What exactly happens when end user makes a request?"

Well answer is definitely "Action method executes". But my exact question is how come controller and action method are identified for a particular URL request.

Before we start with our lab "Implement User Friendly URLs", first let's find out answer for above question. You might be wondering why this topic is coming in the ending. I purposely kept this topic at near end because I wanted people to know MVC well before understanding internals.

Understand RouteTable

In Asp.Net MVC there is a concept called RouteTable which will store URL routes for an application. In simple words it holds a collection defining possible URL patterns of an application.

By default one route will be added to it as a part of project template. To check it open Global.asax file. In Application_Start you will find a statement something like below.

```
RouteConfig.RegisterRoutes(RouteTable.Routes);
```

You will find RouteConfig.cs file inside App_Start folder which contain following code block.

```
namespace WebApplication1
{
    public class RouteConfig
    {
        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 }
            );
        }
    }
}
```

As you can see in RegisterRoutes method already one default route is defined with the help of routes.MapRoute method.

Routes defined inside RegisterRoutes method will be used later in the Asp.Net MVC request cycle to determine the exact controller and action method to be executed.

If it's required we can create more than one route using route.MapRoute function. Internally defining route means creating Route object.

MapRoute function also attach RouteHandler to route object which will be MVCRouteHandler in case of ASP.NET MVC by default.

Understand ASP.NET MVC request cycle

Before you start let me make you clear that we are not going to explain 100% request cycle here. We will touch pass only important ones.

Step 1 – UrlRoutingModule

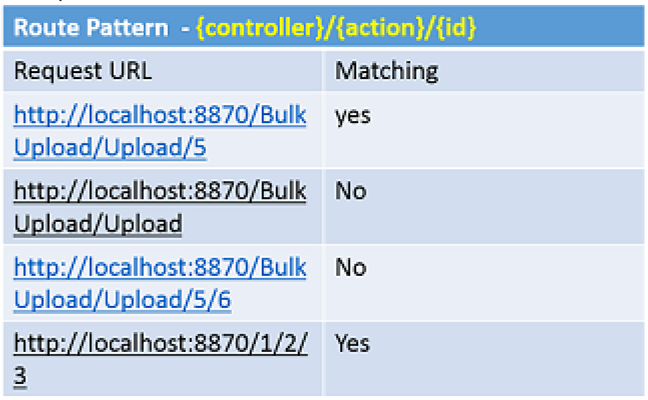
When end user make a request it first pass through UrlRoutingModule Object. UrlRoutingModule is a HTTP Module.

Step 2 – Routing

UrlRoutingModule will get the first matching Route object from the route table collection. Now for matching, request URL will be compared with the URL pattern defined in the route.

Following rules will be considered while matching.

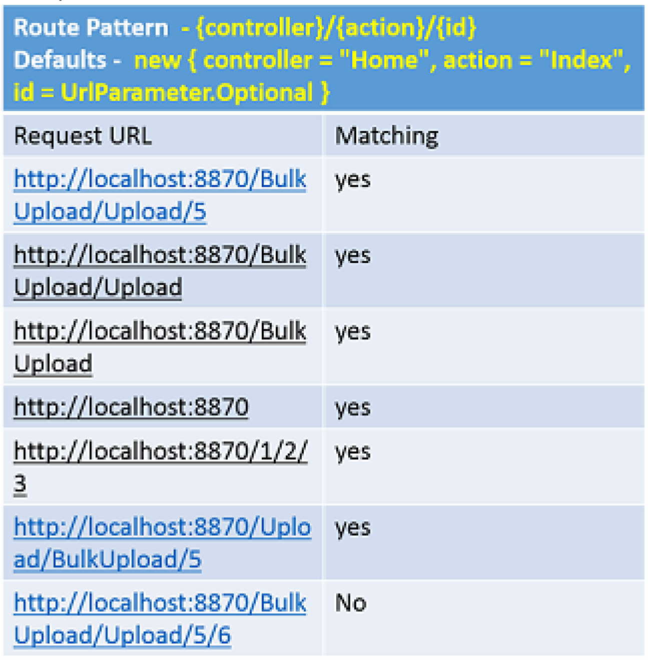
- Number of parameters in the requests URL(other than domain name) and in the URL pattern defined in the route
  Example:



Route Pattern - {controller}/{action}/{id}

Request URL	Matching
http://localhost:8870/BulkUpload/Upload/5	yes
http://localhost:8870/BulkUpload/Upload	No
http://localhost:8870/BulkUpload/Upload/5/6	No
http://localhost:8870/1/2/3	Yes

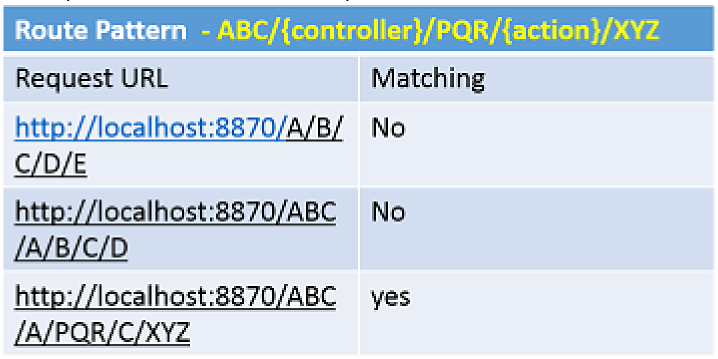
- Optional parameters defined in the URL pattern
  Example:



Route Pattern - {controller}/{action}/{id}
Defaults - new { controller = "Home", action = "Index", id = UrlParameter.Optional }

Request URL	Matching
http://localhost:8870/BulkUpload/Upload/5	yes
http://localhost:8870/BulkUpload/Upload	yes
http://localhost:8870/BulkUpload	yes
http://localhost:8870	yes
http://localhost:8870/1/2/3	yes
http://localhost:8870/Upload/BulkUpload/5	yes
http://localhost:8870/BulkUpload/Upload/5/6	No

- Static parameters defined in the parameter



Route Pattern - ABC/{controller}/PQR/{action}/XYZ

Request URL	Matching
http://localhost:8870/A/B/C/D/E	No
http://localhost:8870/ABC/A/B/C/D	No
http://localhost:8870/ABC/A/PQR/C/XYZ	yes

Step 3 – Create MVC Route Handler

Once the Route object is selected, UrlRoutingModule will obtain MvcRouteHandler object from Route object.

Step 4 – Create RouteData and RequestContext

UrlRoutingModule object will create RouteData using Route object, which it then uses to create RequestContext.

RouteData encapsulates information about routes like name of the controller, name of the action and values of route parameters.

Controller Name

In order to get the controller name from the request URL following simple rule is followed. “In the URL pattern {controller} is the keyword to identify Controller Name”.

Example:

- When URL pattern is {controller}/{action}/{id} and request URL is “http://localhost:8870/BulkUpload/Upload/5”, BulkUpload will be name of the controller.
- When URL pattern is {action}/{controller}/{id} and request URL is “http://localhost:8870/BulkUpload/Upload/5”, Upload will be name of the controller.

Action Method Name

In order to get the action method name from the request URL following simple rule is followed. “In the URL pattern {action} is the keyword to identify action method Name”.

Example:

- When URL pattern is {controller}/{action}/{id} and request URL is “http://localhost:8870/BulkUpload/Upload/5”, Upload will be name of the action method.
- When URL pattern is {action}/{controller}/{id} and request URL is “http://localhost:8870/BulkUpload/Upload/5”, BulkUpload will be name of the action method.

Route Parameters

Basically a URL pattern can contain following four things

1. {controller} -> Identifies controller name
2. {action} -> Identifies actionmethod name.
3. SomeString -> Example – “MyCompany/{controller}/{action}” -> in this pattern “MyCompany” becomes compulsory string.
4. {Something} -> Example – “{controller}/{action}/{id}” -> In this pattern “id” is the route parameter. Route parameter can be used to get the value in the URL itself at the time of request.

Look at the following example.

Route pattern - > “{controller}/{action}/{id}”

Request URL ->http://localhost:8870/BulkUpload/Upload/5

Testing 1

```
public class BulkUploadController : Controller
{
    public ActionResult Upload (string id)
    {
       //value of id will be 5 -> string 5
       ...
    }
}
```

Testing 2

```
public class BulkUploadController : Controller
{
    public ActionResult Upload (int id)
    {
       //value of id will be 5 -> int 5
       ...
    }
}
```

Testing 3

```
public class BulkUploadController : Controller
{
    public ActionResult Upload (string MyId)
    {
        //value of MyId will be null
        ...
    }
}
```

Step 5 – Create MVCHandler

MvcRouteHandler will create the instance of MVCHandler passing RequestContext object.

Step 6 – Create Controller instance

MVCHandler will create Controller instance with the help of ControllerFactory (by default it will be DefaultControllerFactory).

Step 7 – Execute method

MVCHandler will invoke Controller's execute method. Execute method is defined inside controller base class.

Step 8 – Invoke Action method

Every controller is associated with a ControllerActionInvoker object. Inside execute method ControllerActionInvoker object invoke the correct action method.

Step 9 – Execute result.

Action method receives the user input and prepares the appropriate response data and then executes the result by returning a return type. Now that return type may be ViewResult, may be RedirectToRoute Result or may be something else.

Now I believe you have good understanding the concept of Routing so let make our Project URLs more user-friendly with the help of routing.

Lab 31 – Implement User friendly URLs

Step 1 – Redefine RegisterRoutes method

Include additional routes in the RegisterRoutes method as follows.

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

    routes.MapRoute(
    name: "Upload",
    url: "Employee/BulkUpload",
    defaults: new { controller = "BulkUpload", action = "Index" }
    );

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

As you can see now we have more than one route defined.

(Default Route is kept untouched.)

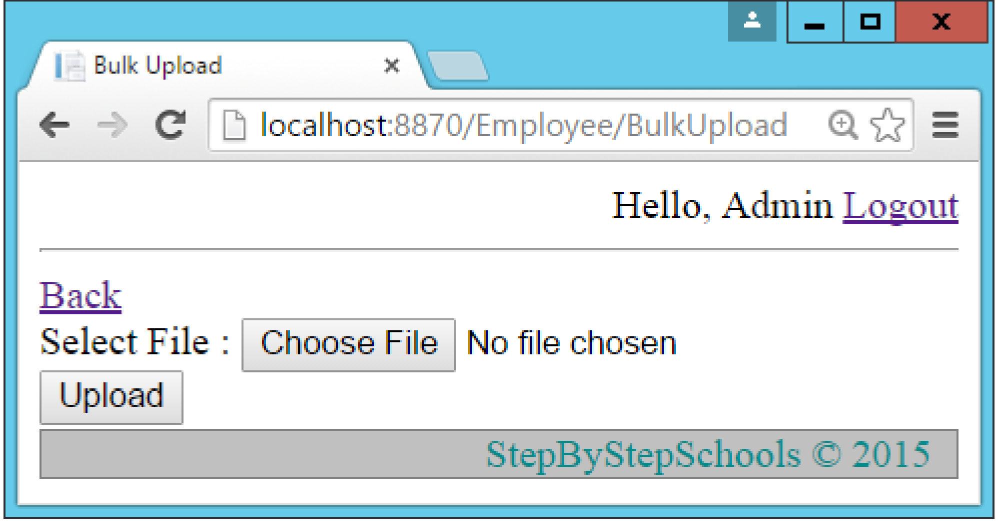
Step 2 – Change URL references

Open AddNewLink.cshtml from "~/Views/Employee" folder and change BulkUpload link as follows.

```
&nbsp;
<a href="/Employee/BulkUpload">BulkUpload</a>
```

Step 3 – Execute and Test

Execute the application and see the magic.


Bulk Upload
localhost:8870/Employee/BulkUpload
Hello, Admin Logout
Back
Select File : Choose File No file chosen
Upload
StepByStepSchools © 2015

As you can see URL is no more in the form of “Controller/Action”. Rather it is more user friendly but output is same.

I recommend you to defines some more routes and try some more URLs.

Talk on Lab 31

Does earlier URL work now?

Yes, earlier URL will work too.

Now Index action in the BulkUploadController is accessible from two URLs

1. “http://localhost:8870/Employee/BulkUpload”
2. “http://localhost:8870/BulkUpload/Index”

What is “id” in default route?

We already spoke about it. It’s called Route Parameter. It can be used to get values via URL. It’s a replacement for Query String.

What is the difference between Route Parameter and Query String?

- Query String have size limitation whereas we can define any number of Route Parameters.
- We cannot add constraints to Query String values but we can add constraints to Route Parameters.
- Default Value for Route Parameter is possible whereas default value for Query String is not possible.
- Query String makes URL cluttered whereas Route Parameter keep it clean.

How to apply constraints to Route Parameter?

It can be done with the help of regular expressions.

Example: look at the following route.

```
routes.MapRoute(
    "MyRoute",
    "Employee/{EmpId}",
    new {controller=" Employee ", action="GetEmployeeById"},
    new { EmpId = @"\d+" }
 );
```

Action method will look like below.

```
public ActionResult GetEmployeeById(int EmpId)
{
   ...
}
```

Now when someone make a request with URL “http://..../Employee/1” or “http://..../Employee/111”, action method will get executed but when someone make a request with URL “http://..../Employee/Sukesh” he/she will get “Resource Not Found” Error.

Is it a required to keep parameter name in action method same as Route Parameter Name?

Basically a single Route pattern may contain one or more RouteParameters involved in it. To identify each route parameter independently it is must to keep parameter name in action method same as Route Parameter Name.

Does sequence while defining custom routes matters?

Yes, it matters. If you remember UrlRoutingModule will take first matching route object.

In the above lab we have defined two routes. One custom route and one default route. Now let say for an instance default route is defined first and custom route is defined second.

In this case when end user make a request with URL "http://.../Employee/BulkUpload" in the comparison phase UrlRoutingModule finds that requested URL matches with the default route pattern and it will consider "Employee" as the controller name and "BulkUpload" as the action method name.

Hence sequence is very important while defining routes. Most generic route should be kept at the end.

Is there an easier way to define URL pattern for Action method?

We can use Attribute based routing for that.

Let's try it.

Step 1 – Enable Attribute based routing.

In RegisterRoutes method keep following line after IgnoreRoute statement.

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

routes.MapMvcAttributeRoutes();

routes.MapRoute(
...
```

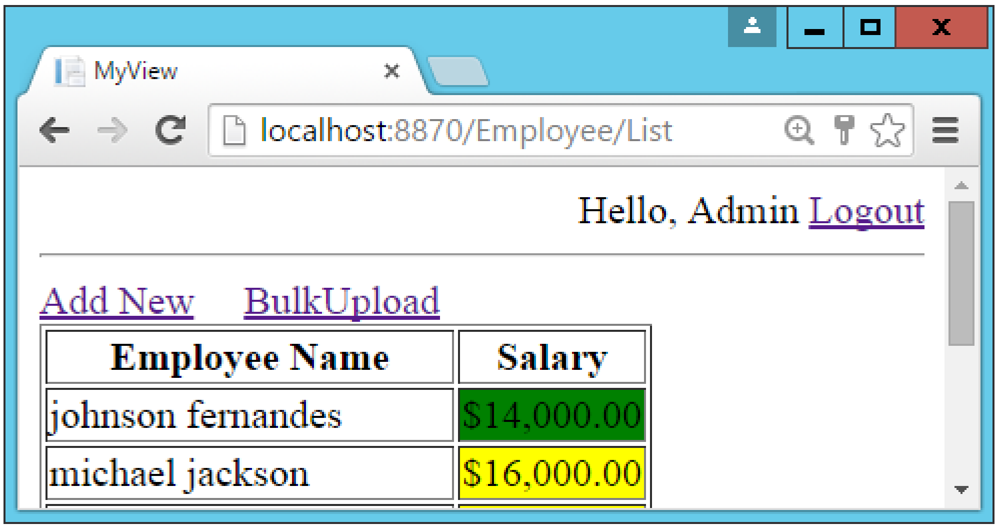
Step 2 – Define route pattern for an action method

Simply attach Route attribute to Action method to Index action of EmployeeController as follows.

```
[Route("Employee/List")]
public ActionResult Index()
{
```

Step 3 - Execute and Test

Execute the application and complete login process.



Hello, Admin Logout

Add New BulkUpload

Employee Name	Salary
johnson fernandes	$14,000.00
michael jackson	$16,000.00

As you can see, we have same output but with different more User Friendly URL.

Can we define Route Parameters with attribute based routing?

Yes, look at the following syntax.

```
[Route("Employee/List/{id}")]
publicActionResult Index (string id) { ... }
```

What about the constraints in this case?

It will be easier.

```
[Route("Employee/List/{id:int}")]
```

We can have following constraints

1. {x:alpha} – string validation
2. {x:bool} – Boolean validation
3. {x:datetime} – Date Time validation
4. {x:decimal} – Decimal validation
5. {x:double} – 64 bit float point value validation
6. {x:float} – 32 bit float point value validation
7. {x:guid} – GUID validation
8. {x:length(6)} –length validation
9. {x:length(1,20)} – Min and Max length validation
10. {x:long} – 64 int validation
11. {x:max(10)} – Max integer number validation
12. {x:maxlength(10)} – Max length validation
13. {x:min(10)} – Min Integer number validation
14. {x:minlength(10)} – Min length validation
15. {x:range(10,50)} – Integer range validation
16. {x:regex(SomeRegularExpression)} – Regular Expression validation

What does IgnoreRoutes do in RegisterRoutes method?

IgnoreRoutes will be used when we don't want to use routing for a particular extension. As a part of MVC template following statement is already written in RegisterRoutes method.

```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
```

It means if end user make a request with an extension ".axd" then there won't be any routing operation. Request will directly reach to physical resource.

We can define our own IgnoreRoute Statement as well.

Conclusion

With Day 6 we have completed our sample MVC project. Hope you have enjoyed the complete series.

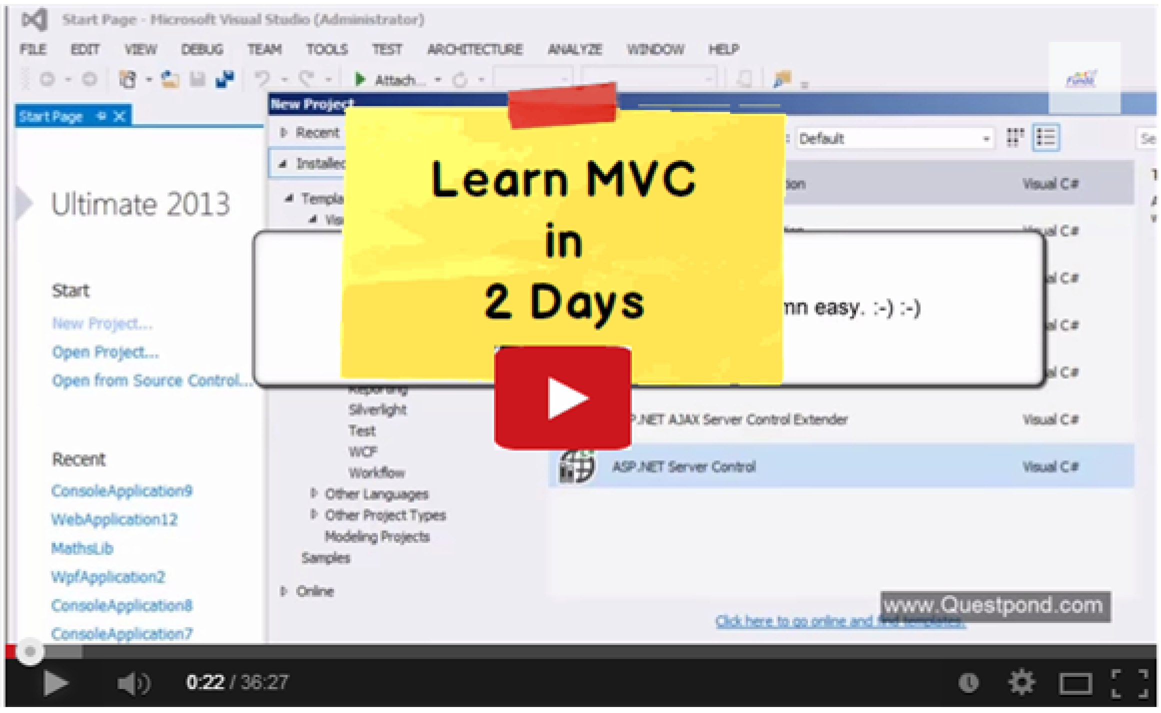
Wait!!! Where is Day 7?

Day 7 will be there my friends. In day 7 we will create a Single Page Application using MVC, jQuery and Ajax. It will be more fun and more challenge.Stay tuned ☺

Your comments, Mails always motivates us do more. Put your thoughts and comments below or send mail to SukeshMarla@Gmail.com

Connect us on Facebook, LinkedIn or twitter to stay updated about new releases.

In case you want to start with MVC 5 start with the below video Learn MVC 5 in 2 days.



License

About the Author

Marla Sukesh

Instructor / Trainer Train IT
India

Learning is fun but teaching is awesome.

Code re-usability is my passion ,Teaching and learning is my hobby, Becoming an successful entrepreneur is my goal.

By profession I am a Corporate Trainer.
I do trainings on WCF, MVC, Business Intelligence, Design Patterns, HTML 5, jQuery, JSON and many more Microsoft and non-Micrsoft technologiees.

Find my profile here

My sites

- **justcompile.com**
- **www.sukesh-marla.com**

@Twitter
@Facebook

Comments and Discussions

50 messages have been posted for this article Visit **https://www.codeproject.com/Articles/1002109/Learn-MVC-Project-in-days-Day** to post and view comments on this article, or click **here** to get a print view with messages.