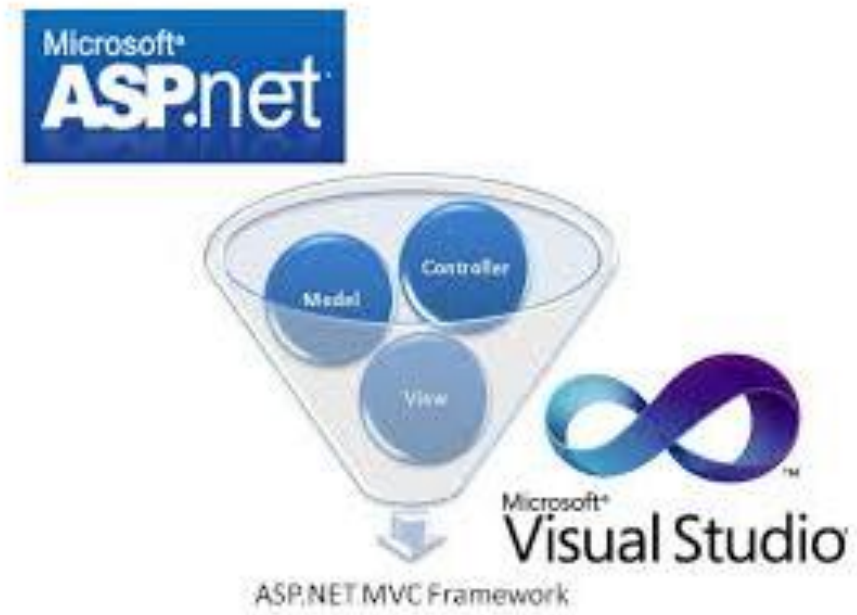


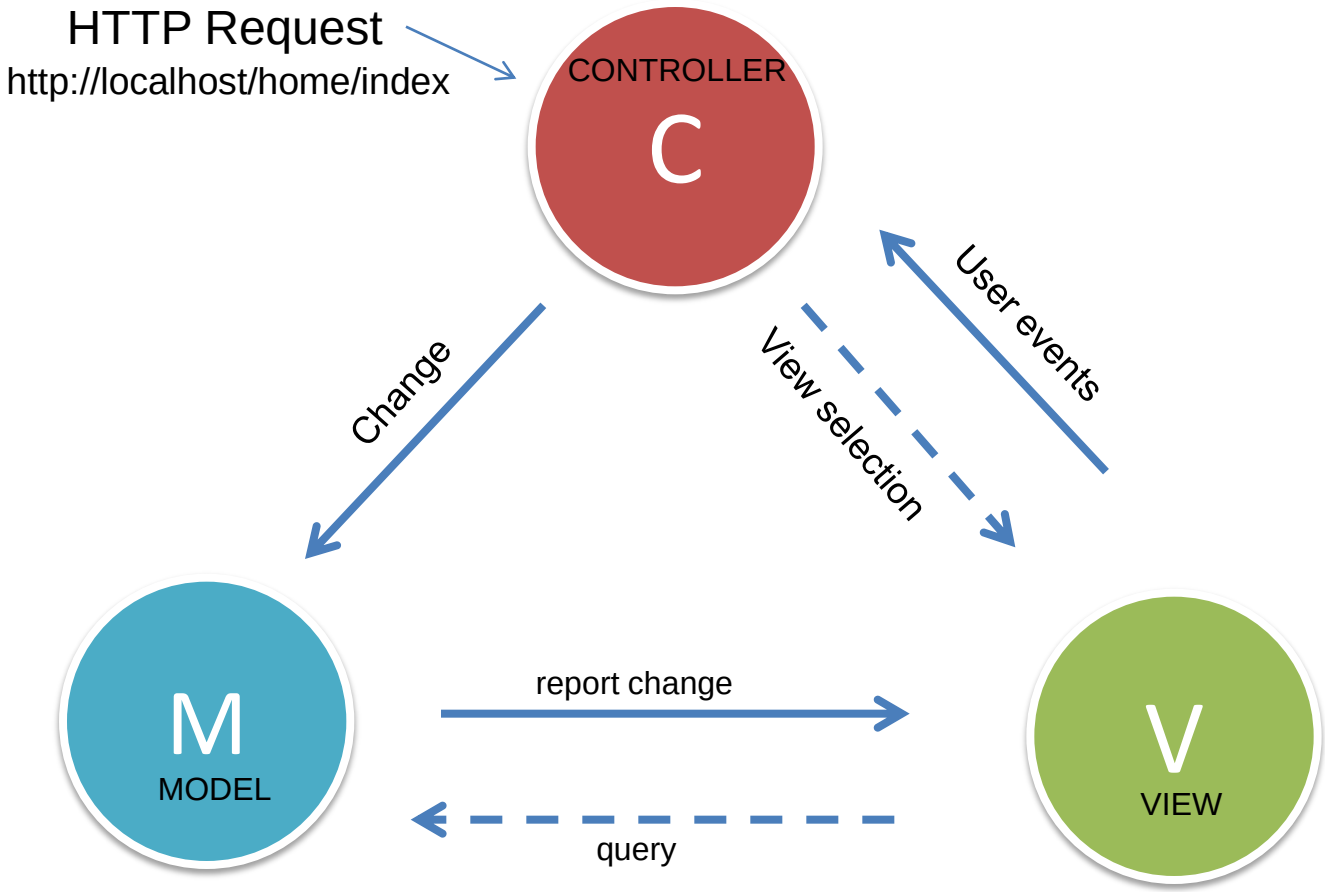
C#

2.MVC CONTROLLERS

Overview

- MVC Controllers
- Routing Rules
- Controller Actions
- Action filters
- Action parameters

MVC CONTROLLER



MVC Controllers

- ASP.NET MVC Framework maps URLs to classes that are referred to as **Controllers**
- **Controllers process incoming requests, handle user inputs and interacts, execute appropriate application logic.**
- **It calls a separate view component to generate the HTML markup for the request.**
- The base class for all controllers is the ControllerBase class, which provides general MVC handling. The Controller class inherits from **ControllerBase**and is the default implement of a controller.
- The **Controller** class is responsible for the following processing stages:
  - Locating the appropriate action method to call and validating that it can be called
  - Getting the values to use as the action method's arguments
  - Handling all errors that might occur during the execution of the action method
  - Providing the default Redering Engine to render views.

Controller Action Methods

```
public class HomeController : Controller
{
    1 reference
    public ActionResult Index()
    {
        return View();
    }

    1 reference
    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";

        return View();
    }

    1 reference
    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";

        return View();
    }
}
```

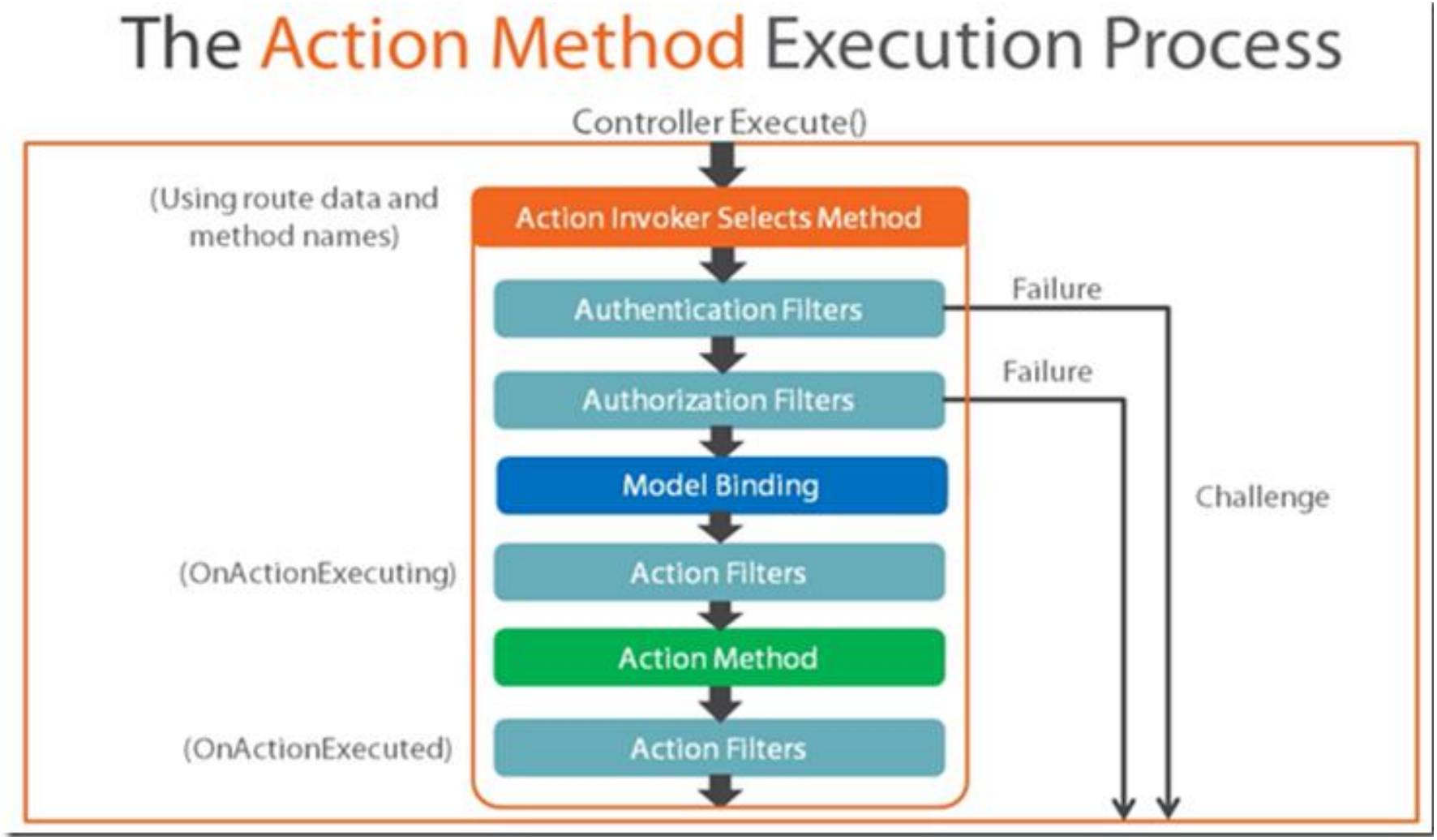
Index() Actionmethod

About() Actionmethod

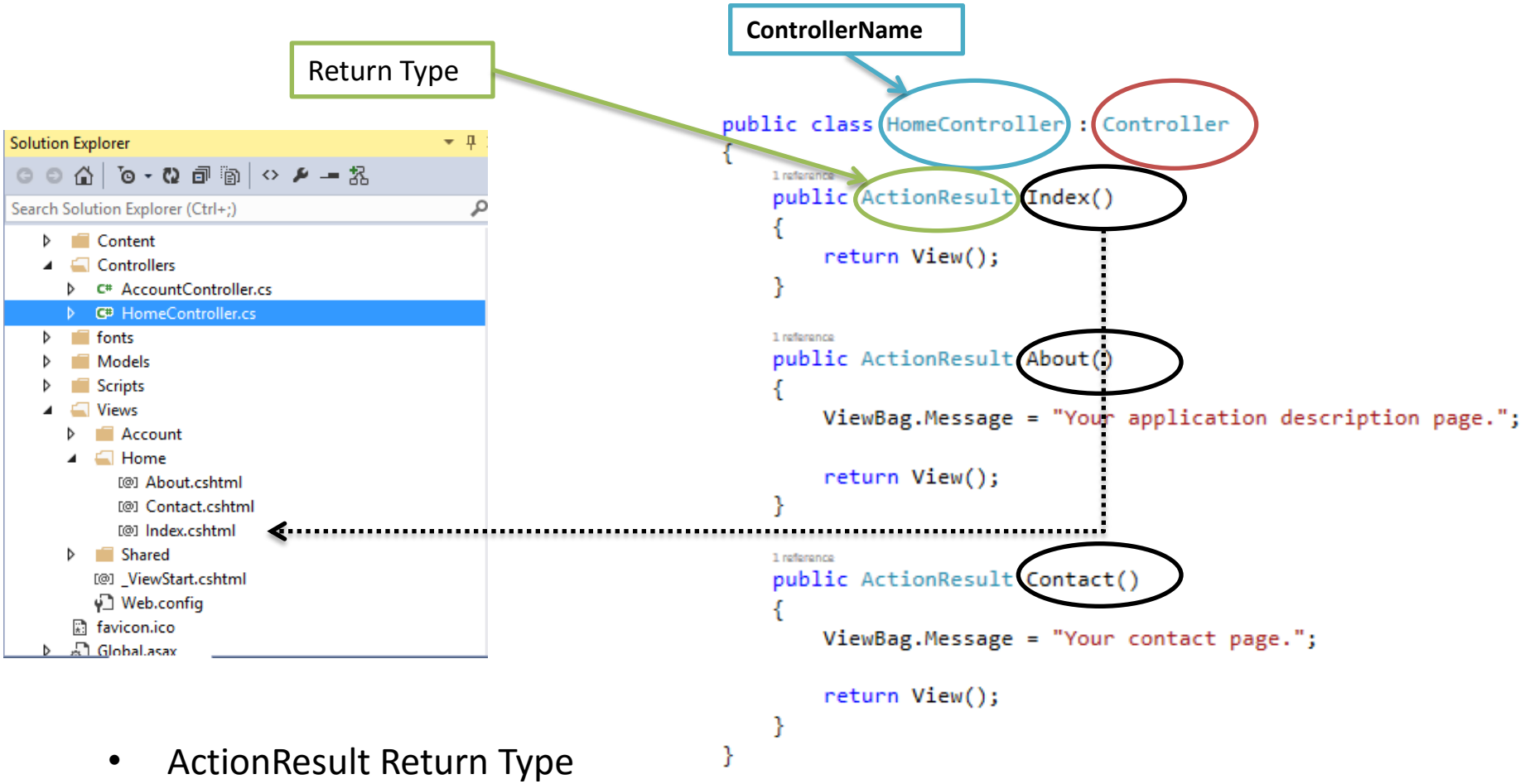
Contact() Actionmethod

- User Interaction with ASP.NET MVC Applications is organized around controllers and action methods.
- The controller defines action methods.
- Controllers can include as many action methods as needed
- **Action methods typically have a one-to-one mapping with user interactions.**
- **On http url request, MVC application uses routing rules that are defined in the GLOBAL.ASAX to parse the URL and determine the path of the Controller**
- **Controller determines appropriate action method to handle the request.**

http://localhost/home/Index -> Index()
http://localhost/home/About -> About()
http://localhost/home/Contact -> Contact()

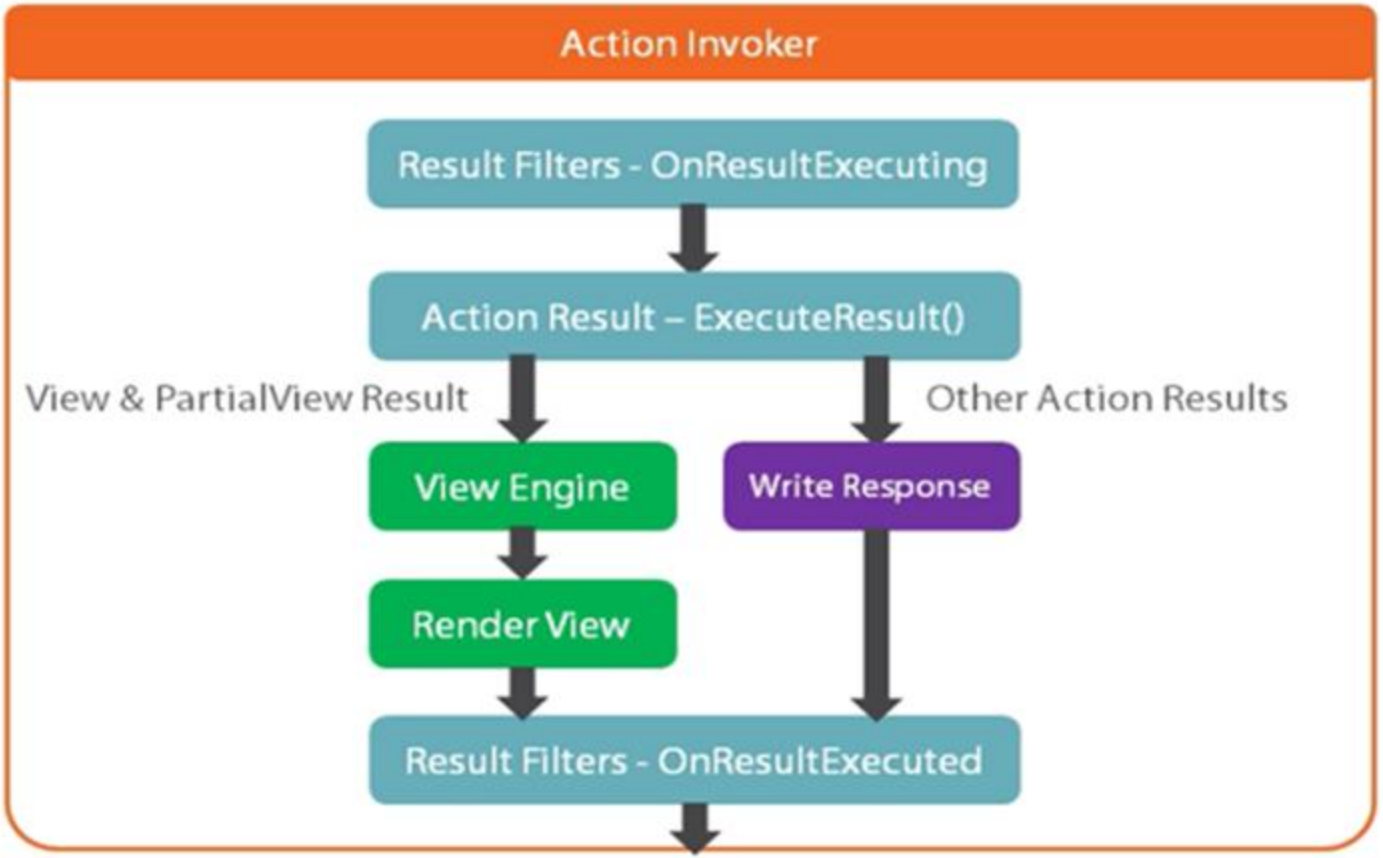


Action Results



- ActionResult Return Type
  - When creating new controllers, they will come with one or more actions by default.
  - The Empty controller includes an Index action with a return value of type **ActionResult**

The Action Result Execution Process



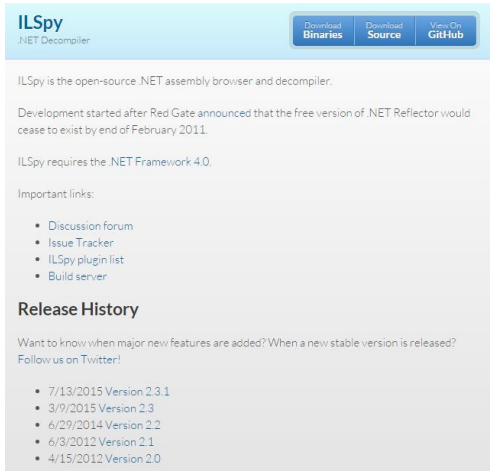
Controller Action Methods

- Actions typically return an ActionResult/VIEWRESULT

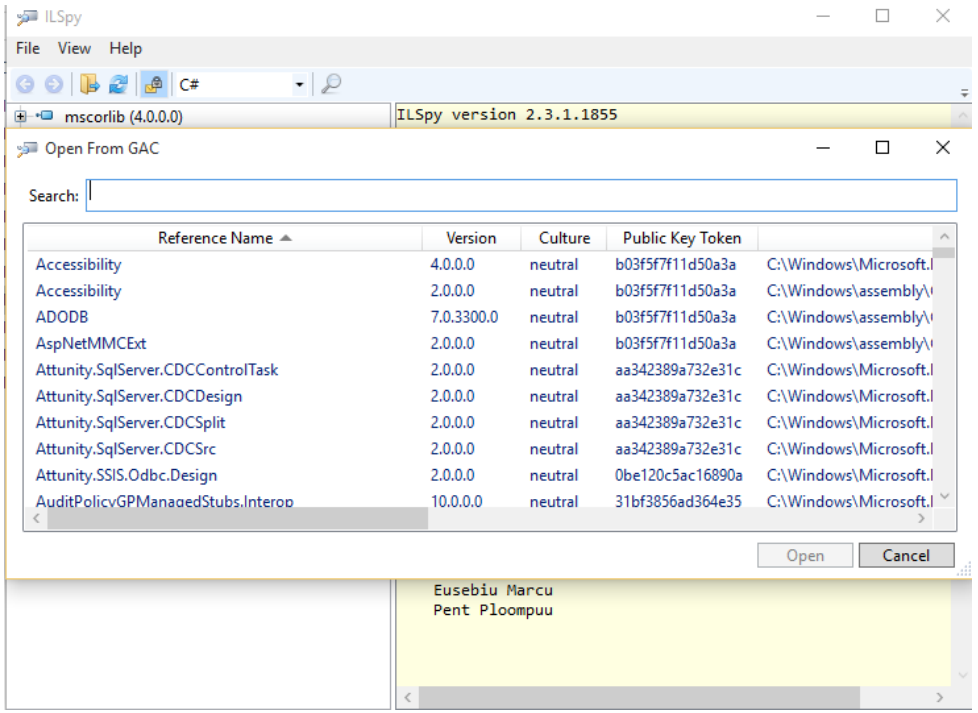
Name	FrameworkBehaviour	Producing Method
ContentResult	Returns a String literal	Content
EmptyResult	No response	
FileContentResult/ FilePathResult/FileStreamResult	Return the contents of a file	File
HttpUnauthorizedResult	Returns an HTTP 403 status	
JavaScript Result	Returns a script to execute	JavaScript
JSONResult	Returns data in JSON format	JSON
RedirectResult	Redirects the client to a new URL	Redirect
RedirectToRouteResult	Redirect to another action, or another controller's action	RedirectToRoute/RedirectToAction
ViewResult PartialViewResult	Response is the responsibility of view engine	View/PartialView

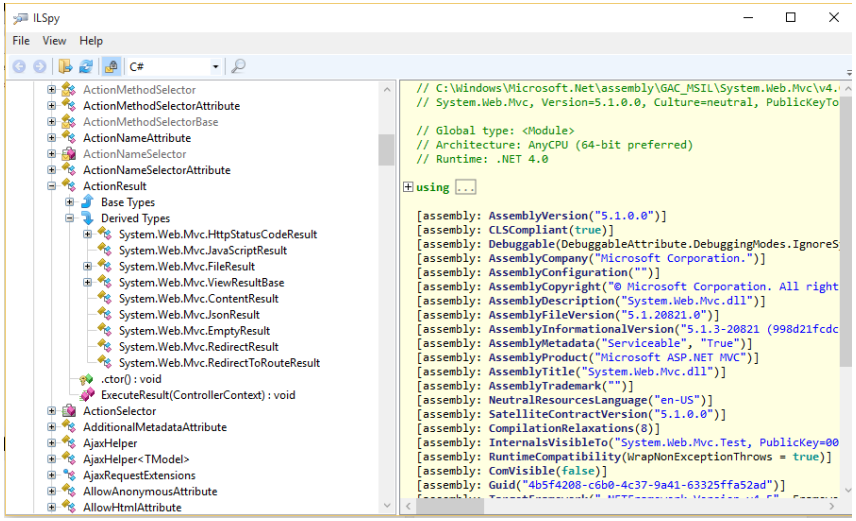
ILSPY

- An opensource .NET assembly browser and decompiler



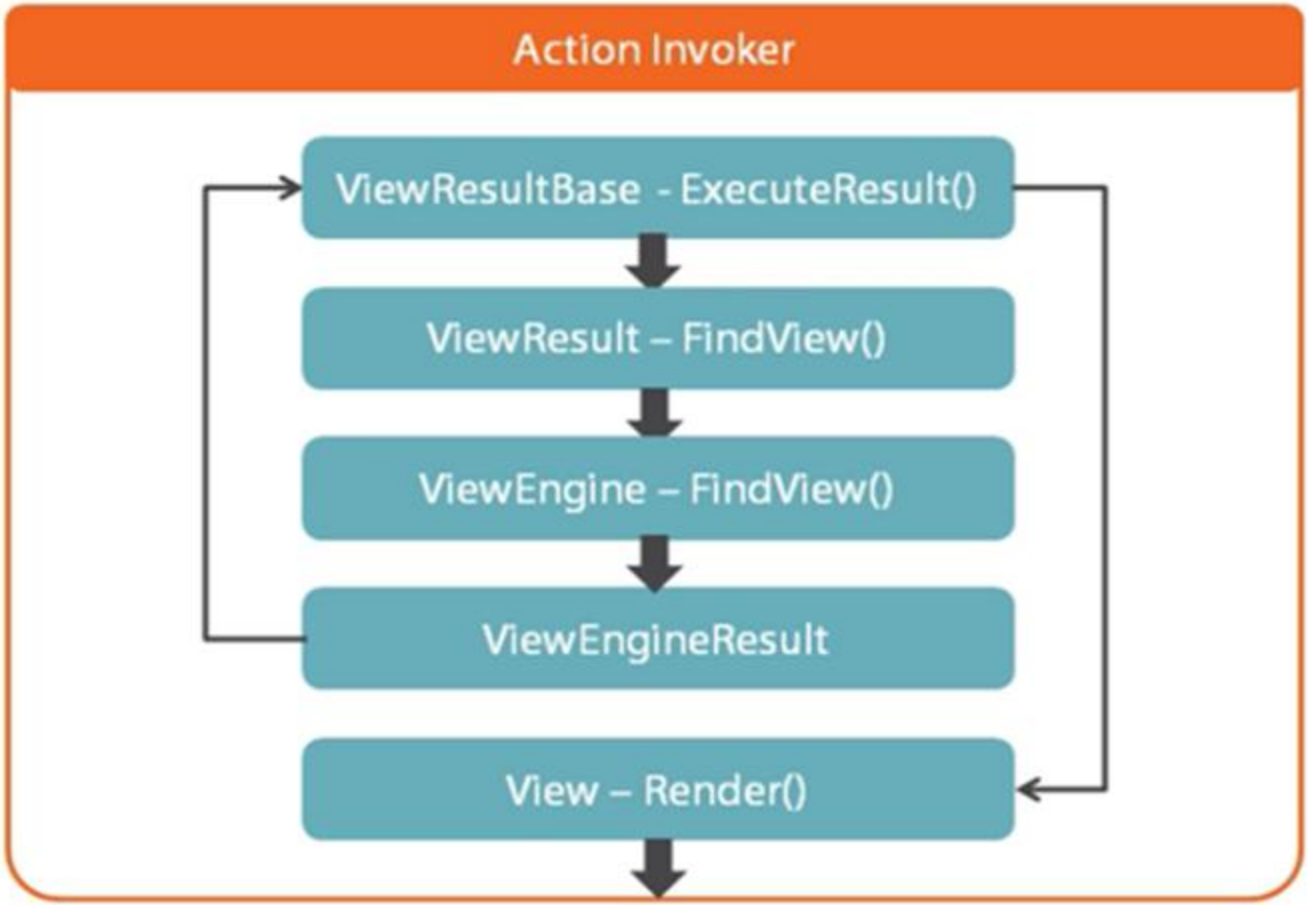
GAC – Global Assembly Cache





- Best Practice
  - Return specific sub-types, but if different paths of the action method return different subtypes, then it is better to return an ActionResult Object

The View Result Execution Process



Action Selectors

- Action selectors are **attributes** that can be applied *to action methods and are used to influence which action method gets invoked in response to a request.*
- **ACTION NAME (ALIAS THE NAME OF ACTION METHOD)**
  - Used to **change the name to invoke an action method.**
- **ACTION VERB**
  - used **when we want to control the selection of the action method based on request type.**

```
[ActionName("Location")]
1 reference
public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";

    return View();
}
```

Using action name

default

http://localhost/home/**location**

http://localhost/home/**contact**

[HttpGet]or [HttpPost] → shortcuts

```
[AcceptVerbs(HttpVerbs.Post)]
1 reference
public ActionResult About()
{
    ViewBag.Message = "Your application description
      page.";

    return View();
}
```

Why use the MVC AcceptVerbs attribute?

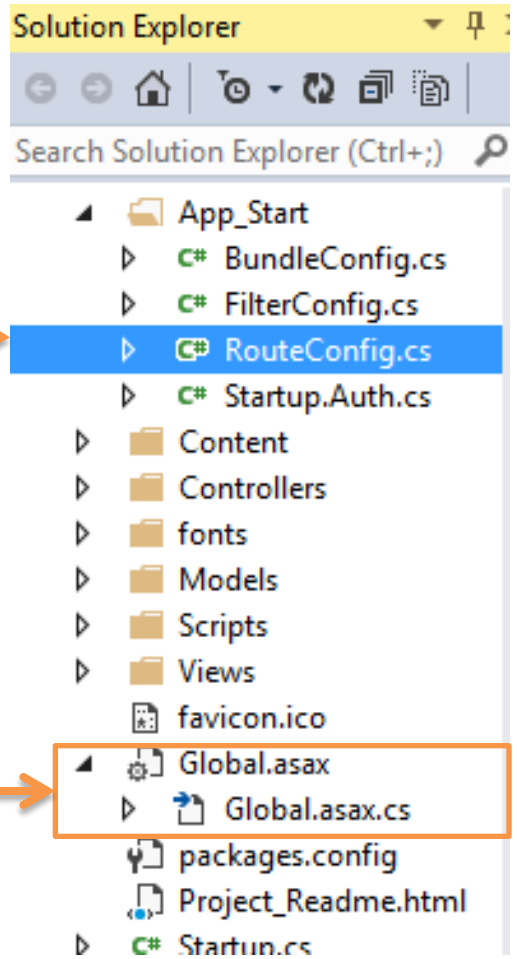
- Can be applied to action methods in a controller so that appropriate overloaded method is invoked for a given request.
- MVC will automatically **dispatch a request to appropriate action method based on the HTTP VERB.**
- Advantage of differentiating methods based on HTTP VERB is that the same URL can be used for multiple purposes (e.g. Display and edit).
  - Which is achieved with 2 separate URLs, but downside is that it makes bookmarking pages difficult.

```
[AcceptVerbs(HttpVerbs.Get)]
public ActionResult Edit(int id) {
   // code snipped
   // this is invoked when viewing the edit page
}

[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Edit(int id, FormCollection formValues) {
   // code snipped
   // this is invoked when POSTing data to the edit page
}
```

Routes and Controllers

RouteData

```
public class RouteConfig
{
    1 reference
    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 }
        );
    }
}
```



Insert this in **Controlleractionmethod**

```
ViewBag.PageBreadCrumbValue = string.Format("{0}>{1}>{2}",
                    RouteData.Values["controller"],
                    RouteData.Values["action"],
                    RouteData.Values["id"]);
```

Define a New Route

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

ACTION FILTERS

- An action filter is an **attribute** that you can **apply to individual controller action – or an entire controller – that modifies the way in which the action is executed.**
- **Users can create their own custom action filters**

```
[ActionName("Location")]
[Authorize(Roles="Admin")]
1 reference
public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";

    return View();
}
```

Action Filter

Name	Description
OutputCache	Cache the output of the controller
ValidateInput	Turn Off request validation and allow dangerous input
Authorize	Restrict an action to authorized users or roles
ValidateAntiForgery Token	Helps prevent cross site request forgeries
HandleError	Can specify a view to render in the event of an unhandled exception

ACTION FILTER

INDIVIDUAL Controller Action Method()

```
[ActionName("Location")]
[Authorize(Roles="Admin")]
1 reference
public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";

    return View();
}
```

ENTIRE CONTROLLER

```
[Authorize(Roles="Admin")]
6 references
public class HomeController : Controller
{
```

GLOBAL

ACTION FILTER TYPES

ASP.NET MVC Framework supports four different types of filters:

If you want to control the order in **which filters of the same type are executed** then you can set a **filter's Order property**.

Authorization filters	**Implements the IAuthorizationFilter Attribute**	used to implement authentication and authorization for controller actions.
Action Filters	Implements the IActionFilter attribute	contains logic that is executed before and after a controller action executes. You can use an action filter, for instance, to modify the view data that a controller action returns.
Result filters	Implements the IResultFilter attribute	Result filters contain logic that is executed before and after a view result is executed. For example, you might want to modify a view result right before the view is rendered to the browser.
Exception Filters	Implements the IExceptionFilter attribute	Exception filters are the last type of filter to run. You can use an exception filter to handle errors raised by either your controller actions or controller action results. You also can use exception filters to log errors.

Filters are executed in the order listed above. For example, authorization filters are always executed before action filters and exception filters are always executed after every other type of filter.

Summary

- Foundational knowledge about how MVC Works
  - MVC Controller
  - Action Methods and Action Parameters
  - Action Names and Action Verbs
  - Routing
  - Action Filters