The `MapRoute()` method, which handles URL routing and mapping, takes three arguments:

- Name of the route (string)
- URL format (string)
- Default settings (object type)

In our case, we named the first route "Default" (which is the route name) and then set the URL as:

```
Controller/action/id
```

The `Controller` here is the name of the controller class. `action` will be the method that needs to be invoked inside that controller class. `id` would be the parameters that need to be passed, if any.

In the default arguments, we create a new object and call it "Home", set the action to Index, and do not pass parameters to it. Note the new anonymous type syntax used to create parameter defaults:

```
    new { controller = "Home", action = "Index", id = "" }
```

The `var` keyword and anonymous types: We normally use classes to wrap behavior and properties, but in C# 3.0, we can create the types anonymously without needing to create classes for them. This can be useful when we need to create light weight classes that have only read-only properties. We can use the anonymous syntax to create those types without the need to create a class for them. We can use the new "`var`" keyword to hold such anonymous types, for example:

```
var ch = new { readOnlyProperty1 = value1,
readOnlyProperty2 = value2 };
```

It is important that we name and assign a value to each of the properties that we are creating. What will be the type of the properties? They will automatically be cast to the data types of the values of the properties specified. The anonymous types will always be derived from the base object class directly. They can only be used within class members and cannot be passed as method arguments (unless they are boxed), return values, or be specified as class-level variables. Once the type is created, it cannot be changed into another type.

So we create a new anonymous type as the last argument of the `MapRoute()` method, passing in variable defaults with three properties, namely controller, action, and parameter.

Now have the `Default.aspx` page under the root directory, which acts as a redirecting page to the main home page of the site (which is `/View/Home/Index.aspx`).

We cannot directly set that as the "default" page since we are using URL routes to process pages instead of using physical files in the URLs.

So in the code-behind of our `Default.aspx` page, we have a simple redirect:

```
public void Page_Load(object sender, System.EventArgs e)
    {
        Response.Redirect("~/Home");
    }
```

So the runtime will first set up routes in the `global.asax` page, then it will process the `Default.aspx` page. Here it faces a redirect to this URL: `/Home`.

The Controller

The MVC framework maps this URL to the route set in the global route table, which currently has only the default one, in this format:

```
Controller/action/id
```

So `/Home` corresponds to a controller named Home, and because we have not specified any action or ID, it takes the default values we specified in the `RegisterRoutes()` method in the `globals.asax.cs`. So the default action was Index and the default parameter was an empty string. The runtime initializes the `HomeController.cs` class, and fires the Index action there:

```
public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ViewData["Title"] = "Home Page";
            ViewData["Message"] = "Welcome to ASP.NET MVC!";
            return View();
        }}
```

In this `Index()` method, we set the data to be displayed in the View (aspx/ascx pages) by using a dictionary property of the base Controller class named `ViewData`. `ViewData`, as the name suggests, is used to set view-specific data in a dictionary object that can hold multiple name/value pairs. When we call the `View()` method, the `ViewData` is passed by the Controller to the View and rendered there.