

Developing Custom Web Services using Web API 2



Agenda

- Understanding API Controllers and Call Routing
- Creating RESTful Web Services
- Custom Routing Schemes and Attribute Routing
- Developing OData Controller
- Cross-Origin Resource Sharing (CORS)



Introducing WebAPI

- Framework and tooling for building RESTful services
- Part of ASP.NET MVC
 - Uses Controller and Routing paradigm
- Tooling, wizards, scaffolding
 - Simplified creation of REST and OData services
 - Simplified use of Entity Framework to wrap database operations
- Can be a stand-alone service or part of an app
 - When added to an app, you perform additional manual modifications to Global.asax



Controllers

- Controllers inherit from `ApiController`

```
public class ValuesController : ApiController
```

- By default methods are mapped to HTTP verbs

```
public IEnumerable<string> Get() {}
```

```
public string Get(int id) {}
```

```
public void Post([FromBody]string value){}
```

```
public void Put(int id, [FromBody]string value){}
```

```
public void Delete(int id){}
```



Routing

- Routes are controlled through maps

```
config.Routes.MapHttpRoute(  
    name: "DefaultApi",  
    routeTemplate: "api/{controller}/{id}",  
    defaults: new { id = RouteParameter.Optional }  
);
```

- Router makes decisions if information is missing
 - Similar to MVC
- By default methods are mapped to HTTP verbs



Responding

■ Content Negotiation is automatic

- accept: "application/json"
- accept: "application/xml"

■ Return IQueryable to support query syntax

```
public IQueryable<string> Get()
{
    var d = new List<string>() {"a", "b" };
    return d.AsQueryable();
}
```

■ Return HttpResponseMessage for headers and status

```
public HttpResponseMessage Get(int id)
{
    return Request.CreateResponse<string>(HttpStatusCode.OK, data[id - 1]);
}
```



Agenda

- ✓ Introducing WebAPI
 - Calling API Controllers from MVC Apps
 - Creating a RESTful Service
 - Creating an OData Service
 - Using Cross-Origin Resource Sharing



Scenarios to Consider

- Additional API Controllers NOT Required
 - You don't require JavaScript access to the data
 - App has direct EF access to database anyway
- Additional API Controllers MAY be Required
 - You require JavaScript access to the data
 - Data source cannot be accessed directly using EF
 - Want to support clients outside of the App



Calling with Managed Code

```
public ActionResult Index()
{
    StringBuilder url = new StringBuilder();
    url.Append(Request.Url.Scheme)
        .Append("://")
        .Append(Request.Url.Host)
        .Append(":")
        .Append(Request.Url.Port)
        .Append("/api/values");

    HttpWebRequest apiRequest = (HttpWebRequest)HttpWebRequest.Create(url.ToString());
    apiRequest.Credentials = CredentialCache.DefaultCredentials;
    apiRequest.Method = "GET";
    apiRequest.Accept = "application/xml";

    HttpResponseMessage apiResponse = (HttpResponseMessage)apiRequest.GetResponse();
    XmlDocument responseDoc = XmlDocument.Load(apiResponse.GetResponseStream());
    XNamespace ns = "http://schemas.microsoft.com/2003/10/Serialization/Arrays";
    List<string> values = (from v in responseDoc.Descendants(ns + "string")
                          select v.Value ).ToList();

    ViewBag.Values = values;
    return View();
}
```



Calling with JavaScript

```
(function () {  
    "use strict";  
  
    jQuery(function () {  
  
        jQuery.ajax({  
            url: "../api/values",  
            type: "GET",  
            headers: {  
                "accept": "application/json",  
            },  
            success: function (data, status, jqXHR) {  
                alert(data[0]);  
            },  
            error: function (jqXHR, status, message) {  
                alert(JSON.stringify(jqXHR));  
            }  
        });  
  
    });  
  
})();
```





DEMO

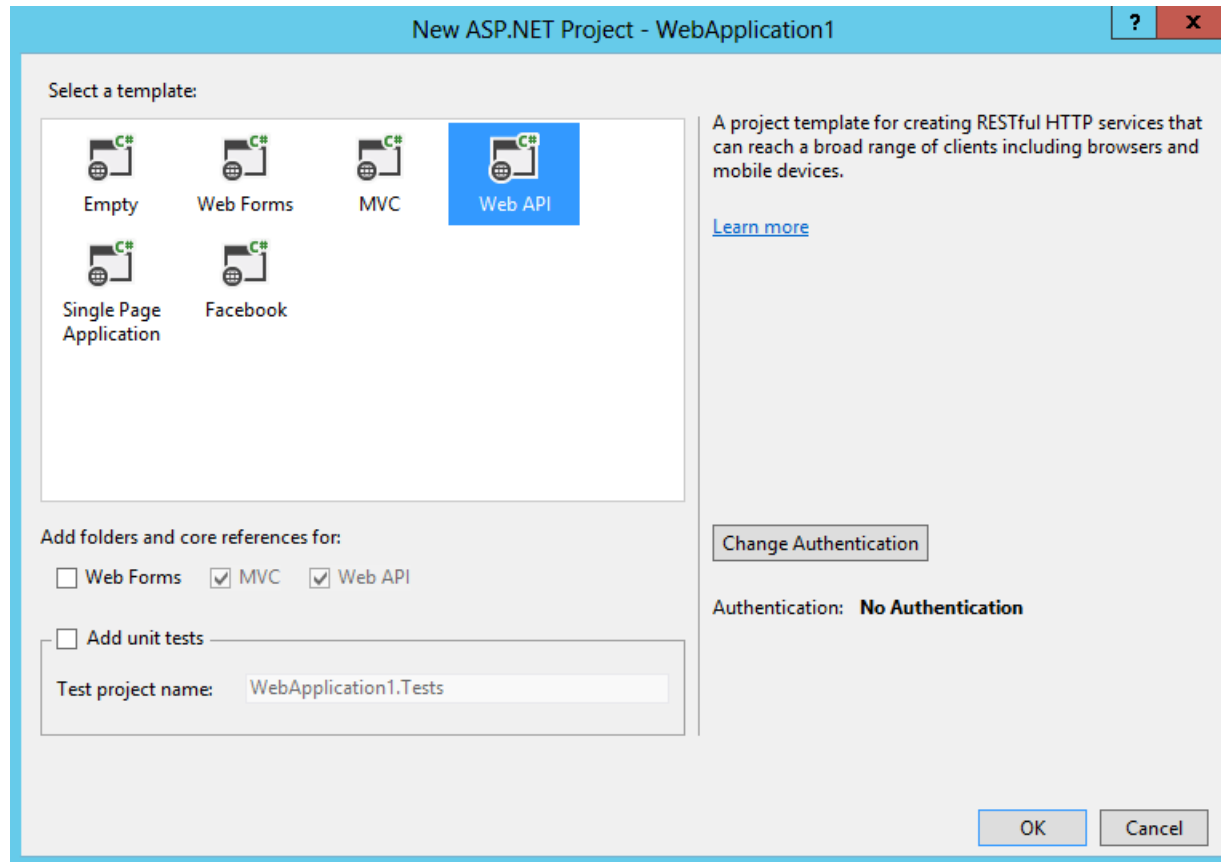
Calling API Controllers from MVC Apps

Agenda

- ✓ Introducing WebAPI
- ✓ Calling API Controllers from MVC Apps
 - Creating a RESTful Service
 - Creating an OData Service
 - Using Cross-Origin Resource Sharing



Creating a Stand-Alone RESTful Service





DEMO

Creating and Testing a RESTful Service

Agenda

- ✓ Introducing WebAPI
- ✓ Calling API Controllers from MVC Apps
- ✓ Creating a RESTful Service
 - Creating an OData Service
 - Using Cross-Origin Resource Sharing



OData Query Options

- \$select
- \$filter
- \$orderby
- \$top
- \$skip
- \$expand



Controllers

- Controllers inherit from `ApiController`
`public class ContactsController : ApiController`
- Methods are mapped to HTTP verbs just like `ApiController`
- Content Negotiation is automatic
- `IQueryable` generated by default



Routing

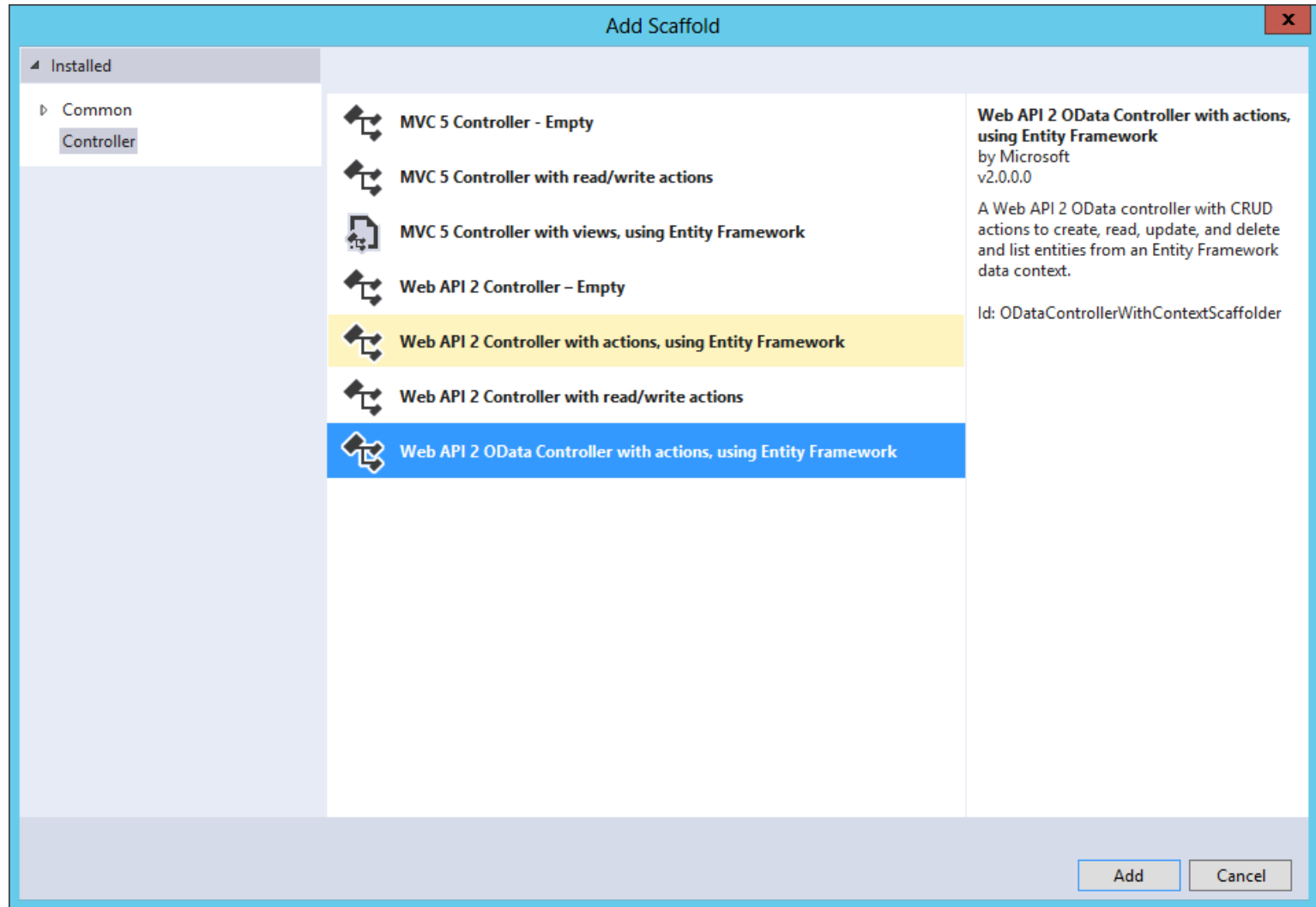
- Routes are controlled through maps

```
odataConventionModelBuilder builder = new ODataConventionModelBuilder();  
builder.EntitySet<Contact>("Contacts");  
builder.EntitySet<Company>("Companies");  
config.Routes.MapODataRoute("odata", "odata", builder.GetEdmModel());
```

- Router makes decisions if information is missing
- By default methods are mapped to HTTP verbs



Adding an OData Controller





DEMO

Creating and Testing an OData Service

Agenda

- ✓ Introducing WebAPI
- ✓ Calling API Controllers from MVC Apps
- ✓ Creating a RESTful Service
- ✓ Creating an OData Service
- Using Cross-Origin Resource Sharing



Cross-Origin Resource Sharing

- Allows JavaScript to make a call across domains
- Superior to JSONP, which only supports GET
- Supported in current versions of all major browsers
- Browser and resource exchange headers
 - Origin header from browser contains origin requesting
 - Access-Control-Allow-Origin header returned from resource if call is allowed
- Enabling in WebAPI2
 - Install Microsoft ASP.NET WebAPI2 CORS NuGet Package
 - Enable CORS in WebApiConfig
 - Use [[EnableCors](#)] attribute in controllers





DEMO

Cross-Origin Resource Sharing

Summary

- ✓ Introducing WebAPI
- ✓ Calling API Controllers from MVC Apps
- ✓ Creating a RESTful Service
- ✓ Creating an OData Service
- ✓ Using Cross-Origin Resource Sharing

