WCF REST

An architectural style is a set of constraints that can be applied when building something.

Roy Thomas Fielding first coined the term REST as a concept in his PhD dissertation.

The constraints of REST are based on the same underlying principles that govern the Web. Those principles are:

User agents interact with resources, and resources are anything that can be named and represented.

Each resource can be addressed via a unique Uniform Resource Identifier (URI).

Interaction with resources (located through their unique URIs) is accomplished using a uniform interface of the HTTP standard verbs (GET, POST, PUT, and DELETE). Also important in the interaction is the declaration of the resource's media type, which is designated using the HTTP Content-Type header. (XHTML, XML, JPG, PNG, and

JSON are some well-known media types.)

Resources are self-descriptive. All the information necessary to process a request on a resource is contained inside the request itself (which allows services to be stateless).

Resources contain links to other resources (hyper-media).

A service that uses the architectural style of REST is generally referred to as a RESTful service or endpoint.

I needed to build a service to work with the data behind MSDN Magazine—a service that could tell me all of the years MSDN Magazine has been published and each of the articles in each issue. Let's say the requirement is that the editors of the magazine could use this service to add new articles and manage the data for upcoming issues.

When building RESTful services, you can go through a very simple set of basic steps to design your service:

1. What are your resources going to be?

2. What are the URIs that are going to be used to represent those resources?

3. What parts of the uniform interface (HTTP verbs) is each URI going to support?

The resources are all of the years that MSDN Magazine has been published, all of the issues published each year, and all of the articles published in each magazine. I'm going to use a media-type application/xml (XML) to represent these resources, but it's important to remember that RESTful services are not in any way limited to XML as the media type.

Next, I need to determine the URIs for each resource. Right now I only need to determine the relative URIs since the absolute URI will be determined by where I host the endpoint. The list of years will be the root URI of the service (/). Using this syntax, /{year} will return all of the issues for each year; /{year}/{issue} will be the URI for each issue (I'll identify each issue by its month of publication); and /{year}/{issue}/{article} will represent each article (I'll assume each article is numbered from 1 to n in each issue).

Next comes the mapping of URIs to the uniform interface.

Since the magazine's history should really be read-only, the root resource will only expose GET.

A new year can be added by doing a PUT to the URI /{year}. PUT is used to create new resources when the URI of the new resource is known by the client, as it would be in this case. PUT can also be used to update existing resources when the URI is known.

POST is used to create a resource when the URI of the new resource isn't known by the client, so POST will be the verb I'll use when adding a new article resource, which would be sent to the /{year}/{issue} URI.

|  |  |  |
| --- | --- | --- |
| Resource | URI | Verbs |
| All years | "/ " | GET |
| A particular year's issues | "/{year}" | GET, PUT |
| A particular issue | "/{year}/{issue}" | GET, PUT |
| An article | "/{year}/{issue}/{article}" | GET, POST\* |

\* (the article number will be assigned by the system), PUT, DELETE (delete would be turned off once an issue has been published)

WCF is the Microsoft framework for building applications that communicate over a network, regardless of the style or protocol. The programming model centers around two new attributes, WebGetAttribute and WebInvokeAttribute, and a URI template mechanism that enables you to declare the URI and verb to which each method is going to respond.

WCF simplifies building connected systems is by routing network messages to methods on instances of the classes you define as implementations of your service. The value of Action is based either on the name of your method (plus the namespace of your service) or a custom value (set via the OperationContractAttribute.Action property).

WebGetAttribute tells the dispatcher that the method should respond to HTTP GET requests. WebInvokeAttribute is mapped to HTTP POST by default, but the WebInvokeAttribute.Method property can be set to support any of the other HTTP verbs (PUT and DELETE being the two most common). By default, the URI is determined by the name of the method (added onto the base URI of the endpoint).

WCF added the ability to define the URI for each resource by using a special template syntax, such as the one I used earlier in this column for describing the MSDN Magazine service endpoint. This syntax allows you to define, with replaceable tokens, the URI structure you'd like each method to represent in conjunction with the HTTP verb (via the WebGetAttribute or WebInvokeAttribute).

[ServiceContract]

public interface IMSDNMagazineService

{

[OperationContract]

[WebGet(UriTemplate="/")]

IssuesCollection GetAllIssues();

[OperationContract]

[WebGet(UriTemplate = "/{year}")]

IssuesData GetIssuesByYear(string year);

[OperationContract]

[WebGet(UriTemplate = "/{year}/{issue}")]

Articles GetIssue(string year, string issue);

[OperationContract]

[WebGet(UriTemplate = "/{year}/{issue}/{article}")]

Article GetArticle(string year, string issue, string article);

[OperationContract]

[WebInvoke(UriTemplate = "/{year}/{issue}",Method="POST")]

Article AddArticle(string year, string issue, Article article);

}

Notice that the UriTemplate syntax allows for multiple variable path segments, and that each of those path segments are passed to the methods as arguments.

In WCF, a binding determines how WCF is going to communicate. A binding is really the configuration that tells WCF how to build what is known as the channel stack, which is the set of objects that will work together to provide the type of communication you want for a particular endpoint. For a RESTful endpoint, the binding you use is WebHttpBinding.

ServiceHost sh =

new ServiceHost(typeof(MSDNMagazineServiceType));

string baseUri = "<http://localhost/MagazineService>";

ServiceEndpoint se =

sh.AddServiceEndpoint(typeof(IMSDNMagazineService),

new WebHttpBinding(), baseUri);

se.Behaviors.Add(new WebHttpBehavior());

sh.Open();

Of course, I can do this with configuration as well.

<configuration>

<system.serviceModel>

<services>

<service name="MSDNMagazine.MSDNMagazineServiceType">

<endpoint

address="<http://localhost/MagazineService>"

binding="webHttpBinding"

contract="MSDNMagazine.IMSDNMagazineService"

behaviorConfiguration="webby"/>

</service>

</services>

<behaviors>

<endpointBehaviors>

<behavior name="webby">

<webHttp/>

</behavior>

</endpointBehaviors>

</behaviors>

</system.serviceModel>

</configuration>

WebServiceHost is a ServiceHost-derived type, which simplifies self-hosting scenarios of RESTful endpoints.

string baseUri = "<http://localhost/MagazineService>";

WebServiceHost sh =

new WebServiceHost(typeof(MSDNMagazineServiceType),

new Uri(baseUri));

sh.Open();

To simplify the managed hosting scenario, Microsoft added WebServiceHostFactory, which uses an open WCF extensibility point (using a custom ServiceHostFactory type) in the managed hosting scenario to create a configuration-free experience for many RESTful services. The .svc file looks like this:

<%@ ServiceHost Factory= "System.ServiceModel.Activation.WebServiceHostFactory" Service="MSDNMagazine.MSDNMagazineServiceType" %>

WebServiceHostFactory creates an instance of the WebServiceHost, and since the WebServiceHost will auto-configure the endpoint using WebHttpBinding and WebHttpBehavior, there doesn't need to be any configuration for this endpoint in the web.config at all.

If I did need to customize the binding, I could still add the appropriate entries in the configuration file.

<configuration>

<system.serviceModel>

<services>

<service name="MSDNMagazine.MSDNMagazineServiceType">

<endpoint

address="<http://localhost/MagazineService>"

binding="webHttpBinding"

contract="MSDNMagazine.IMSDNMagazineService"

behaviorConfiguration="webby"/>

</service>

</services>

<bindings>

<webHttpBinding>

<binding name="secure">

<security mode="Transport">

<transport clientCredentialType="Basic"/>

</security>

</binding>

</webHttpBinding>

</bindings>

<behaviors>

<endpointBehaviors>

<behavior name="webby">

<webHttp/>

</behavior>

</endpointBehaviors>

</behaviors>

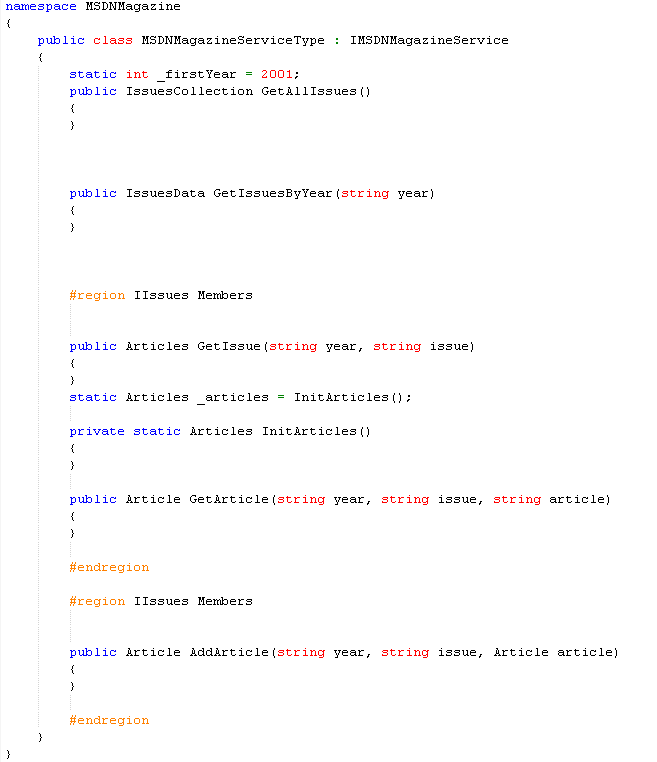
</system.serviceModel>

</configuration>

The REST web service:

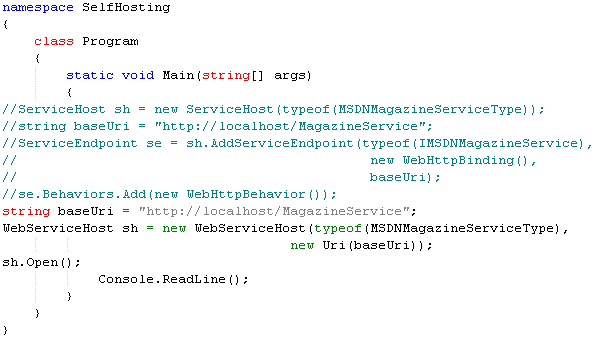
<%@ ServiceHost Factory="System.ServiceModel.Activation.WebServiceHostFactory" Service="MSDNMagazine.MSDNMagazineServiceType" %>





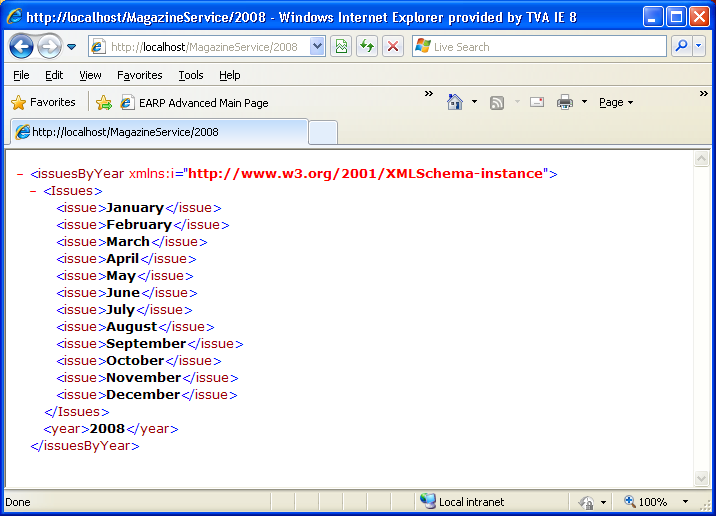
To test the REST web service:





Start the service from VS. You can enter the following:

<http://localhost/MagazineService/2008>



<http://localhost/MagazineService/2008/June>

