

Lullaby

User Guide

Contents

Overview	3
Getting Started	4
Attributes	6
Configuration Section.....	8
Authentication.....	11
Caching.....	13
Serialization	15
Client Library.....	16

Overview

Lullaby is a highly extensible attribute-based framework built on .NET Framework 3.5 SP1 for creating RESTful .NET services.

Requirements and Dependencies

The following requirements and dependencies are required:

- .NET Framework 3.5 SP1
- Json.NET (for JSON serialization)
- System.Web.Routing (for service URL routing)

About

Lullaby was created by Ryan Olshan to provide an alternative to WCF REST and provide features and extensibility that existing .NET REST frameworks lacked.

- Google Code Project – <http://code.google.com/p/ryanolshanrest>
- Google Group – <http://groups.google.com/group/lullaby-rest>
- Google Group Email – lullaby-rest@googlegroups.com

Getting Started

1. Add a reference to Lullaby.dll.
2. Add a reference to System.Web.Routing.dll if using service URL routing. Otherwise, skip to step 3.
3. Create an ASHX handler that inherits from `Lullaby.RestHandler` or configure service URL routing as follows:
 - a. Register the `System.Web.Routing.UrlRoutingModule` HTTP module in Web.config in the httpModules section of system.web or the modules section of system.webServer if using IIS 7.

```
<add name="Routing"
      type="System.Web.Routing.UrlRoutingModule,
      System.Web.Routing, Version=3.5.0.0, Culture=neutral,
      PublicKeyToken=31BF3856AD364E35" />
```

- b. In Global.asax, register a route for the REST service in the ApplicationStart event.

```
public void Application_Start(object sender, EventArgs e)
{
    var route = new Route("myservice/{*path}"), new
    RestRouteHandler("myservice");
    RouteTable.Routes.Add(route);
}
```

4. Mark the assembly with the `RestAssembly` attribute. This attribute is required to make the assembly discoverable by default.

```
[assembly: RestAssembly]
```

5. Create a class and mark it with the `RestClass` attribute.
6. Mark methods within the REST class that should be exposed through the REST service with the `RestMethod` attribute. A REST method needs to be public to be discovered.

```
[RestClass]
public class MyRestClass
{
    [RestMethod("{addressID}", HttpMethod.Method.GET)]
    public Address GetAddress(addressID)
    {
        // Code here
    }
}
```

In the above example, if the URL of the REST service was `http://mysite.com/API`, the above GET method would be accessible via `http://mysite.com/API/MyRestClass/12` to invoke the `GetAddress` method with a value of 12 for the `addressID` parameter.

Note: The URI template placeholders need to match the name of a methods parameter. For example, to match a parameter of `personID`, the URI template placeholder needs to be `{personID}`.

Attributes

CacheInvalidationMatchAttribute – The **CacheInvalidationMatch** attribute is used for specifying matches for cache invalidation. Multiple **CacheInvalidationMatch** attributes can be added to a method.

- **ResourceTemplate** – Template used for matching request URIs. The **ResourceTemplate** can contain format placeholders, which are formatted using the **ParameterNames** property (i.e. `/Address/{0}`), and regular expressions.
- **ParameterNames (Optional)** – Names of the methods parameters that should be used for formatting the **ResourceTemplate** with the value of the parameter. If the parameters type is a non-primitive .NET type that is deserialized by Lullaby from the request stream, the **RestPrimaryKey** attribute needs to be used on a property of the REST class that will be substituted for the placeholder's value. Multiple values for this property should in the form of a comma separated list (i.e. *addressID, name, description*).
- **UseRegularExpressions (Optional)** – Set this property to true to enable the matching of regular expressions on the **ResourceTemplate**.

Note: In order for caching to work properly, the **CacheInvalidationMatch** attribute needs to be used on REST methods that respond to DELETE and PUT method requests so Lullaby knows which URLs to invalidate in the cache when a resource is updated or deleted. Otherwise, the cache will not expire until the configured or default time has passed thus creating the potential for stale items in the cache.

RequireAuthenticationAttribute – The **RequireAuthentication** attribute is used to require authentication on a class or method. When used on a class, it will require authentication on all REST methods.

RestAssemblyAttribute – The **RestAssembly** attribute is used to mark an assembly for default REST assembly auto-discovery by Lullaby.

RestClassAttribute – The **RestClass** attribute is used to mark a class as a REST class.

- **DeserializationType** – The type to deserialize the request stream as. If a value is provided for this property, it will deserialize the request stream of all REST methods in the class to this type.
- **ExcludeFromCache** – This property determines if REST methods contained in a class should be excluded from caching.
- **IgnoreClassName** – This property determines if the class' name shouldn't be appended to the request URL and is false by default. If a class' name were *Address*, setting this property to true would cause the request URL to be `http://mysite.com/API/[request here]` instead of `http://mysite.com/API/Address/[request here]`.

- **Name** – This property sets the class name that is rendered in the request URL. If left blank, it will be rendered as the class's name.

RestMethodAttribute – The `RestMethod` attribute is used to mark a method as a REST method. Multiple `RestMethod` attributes can be added to a method.

- **DeserializeRequestStream** – This property determines if the request stream should be serialized. It is set to false by default for GET and DELETE methods and true for POST and PUT methods.
- **DeserializationType** – The type to deserialize the request stream as. If a value is not provided for this property, the request stream will be deserialized as the REST methods enclosing type.
- **ExcludeFromCache** – This property determines if the method should be excluded from caching.
- **RequestType** – The HTTP method to listen for. Valid options are `HttpMethods.Method.DELETE`, `HttpMethods.Method.GET`, `HttpMethods.Method.POST`, and `HttpMethods.Method.PUT`.
- **UriTemplate** – The URI template to use for matching request URLs to REST methods. For more information on URI templates, see <http://msdn.microsoft.com/en-us/library/system.uritemplate.aspx>.

RestPrimaryKeyAttribute – The `RestPrimaryKey` attribute is used on a property of a REST class to specify that the property's value should be returned when formatting a placeholder for cache invalidation of a non-primitive .NET type that is deserialized from the request stream.

Configuration Section

Lullaby runs out of the box with no configuration settings needed in Web.config other than setting up an ASHX handler or registering `System.Web.Routing.UrlRoutingModule` as an HTTP module. The following settings are available to extend Lullaby. Before Lullaby can be extended, the `RestConfigurationSection` configuration section needs to be set in Web.config.

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="restSettings"
type="Lullaby.ConfigurationHandler.RestConfigurationSection, Lullaby"
/>
  </configSections>
  <restSettings>
    ...
  </restSettings>
</configuration>
```

Authentication – The authentication element is used for specifying settings related to Lullaby's authentication feature.

```
<authentication
  providerType=""
  tokenExpiration=""
  useIsOrFormsAuthentication="[true|false]" />
```

- `providerType` – The fully qualified type name of the authentication provider.
- `tokenExpiration` – The time in minutes that a security token is valid for. The default value is 60 minutes.
- `useIsOrFormsAuthentication` – If set to true, IIS controlled (basic, integrated, NTLM, etc.) or forms authentication will be used for authentication. This is set to false by default.

Caching – The caching element is used for specifying settings related to Lullaby's caching framework.

```
<caching
  cacheDuration=""
  cacheInvalidationFileNameOverride=""
  cacheInvalidationFilePathOverride=""
  disable="[true|false]"
  fileCacheProviderFileLocation=""
  ignoreCacheInvalidationMatchAttributes="[true|false]"
  providerType="" />
```

- `cacheDuration` – The time in minutes that the cache is valid for. The default value is 60 minutes.

- `cacheInvalidationFileNameOverride` – File name of the cache invalidation XML file. By default, this value is `CacheInvalidation.xml`.
- `cacheInvalidationFilePathOverride` – Directory where the cache invalidation file will reside. By default, Lullaby expects the cache invalidation file to reside in the `websites bin` directory.
- `disable` – Setting this to `true` will disable caching.
- `fileCacheProviderFileLocation` – When using the built-in file cache provider, a value will need to be set to indicate the file path where the cache should be stored on disk.
- `ignoreCacheInvalidationMatchAttributes` – Setting this to `true` will ignore `CacheInvalidationMatch` attributes on REST methods. Since by default cache invalidation matches are generated off of both the cache invalidation XML file, if present, and `CacheInvalidationMatch` attributes, this option allows those attributes to be ignored so only cache invalidation matches specified in the cache invalidation XML file are used.
- `providerType` – The fully qualified type name of the cache provider.

Note: By default, caching is disabled until caching is setup in `Web.config` because cache invalidation needs to be configured in order for caching to work properly. Otherwise, items in cache won't be invalidated against DELETE and PUT methods and will expire when the configured or default cache expiration passes, thus creating the potential for stale items in cache.

RestAssemblies – Collection of assemblies containing REST classes and methods that should be used to search for URI template matches in.

```
<restAssemblies>
  <add assembly="MyAssembly1" />
  <add assembly="MyAssembly2" />
</restAssemblies>
```

- `assembly` - Fully qualified name of the assembly. The assembly can be an assembly loaded in the application domain or Global Assembly Cache.

RestAssemblyDirectories – Collection of directories containing assemblies with REST classes and methods that should be used to search for URI template matches in.

```
<restAssemblyDirectories>
  <add directory="c:\MyDirectory1" searchSubFolders="false" />
  <add directory="c:\MyDirectory2" searchSubFolders="true" />
</restAssemblyDirectories>
```

- `directory` – The path to the directory containing assemblies.
- `searchSubFolders` – Boolean indicating if sub folders within the folder should be searched for assemblies.

Serialization – The serializer to use to serialize and deserialize data for GET methods if no content type is specified and for serializing and deserializing data to and from cache. If no value is provided, the `XmlSerializer` will be used. Lullaby has two built-in serializers: `Lullaby.Serialization.JsonSerializer` and `Lullaby.Serialization.XmlSerializer`.

```
<serialization>
  <defaultSerializer="Lullaby.Serialization.XmlSerializer, Lullaby"
/>
</serialization>
```

- `defaultSerializer`– The fully qualified type name of the serializer to use for serialization and deserialization. The serializer needs to implement `Lullaby.Serialization.ISerializer`.

Authentication

Lullaby has the capability to use IIS controlled authentication (basic, integrated, NTLM, etc.) or authentication using its authentication provider framework. To use the authentication provider framework, an authentication provider needs to be created that implements `IAuthenticationProvider`. The provider also needs to be setup in the authentication section of `restSettings` in `Web.config`.

The following request headers are available for authentication:

- Lullaby-API-Key – API key, password, etc. that will be used to authenticate against a username.
- Lullaby-API-Username – Username to authenticate against.
- Lullaby-Authenticate-As (Optional) – Used for authenticating as another user.
- Lullaby-Security-Token (Optional) – Used for authenticating via a security token that is issued on a successful login attempt. If the security token has expired or is not valid, authenticating will default to using the authentication provider.

The following response headers are available for authentication:

- Lullaby-Authenticated-As – The user that was authenticated as.
- Lullaby-Security-Token – A unique identifier that is created for the purposes of identification without having to re-authenticate using the authentication provider.
- Lullaby-Security-Token-Expiration – Date and time in UTC that the security token will expire.

Below is an example of an authentication provider.

```
using Lullaby.Providers.Authenticaiton;

public class MyAuthenticationProvider : IAuthenticationProvider
{
    public string AuthenticationType
    {
        get { return "My authentication provider"; }
    }

    public bool Authenticate(string username, string key)
    {
        // Authentication code here
    }

    public bool CanImpersonateUser(string
userNameToImpersonate, string userName, string key)
    {
        //Code here to determine if user can impersonate user
    }
}
```

```
public string[] GetUserRoles(string username)
{
    // Return a users roles
}
}
```

- **AuthenticationType** – The value of this property is used to populate the **AuthenticationType** of the **Identity** of the current **HttpContext**'s **User** object.
- **Authenticate** – **Authenticate** authenticates a user with the given username and key. The authenticated username is stored on the **Name** property of the **Identity** of the current **HttpContext**'s **User** object.
- **CanImpersonateUser** – **CanImpersonateUser** determines if a user can impersonate a user. If this feature isn't going to be implemented, return false.
- **GetUserRoles** – Gets the roles assigned to the specified user. If this feature isn't going to be implemented, return null or an empty string array. User roles are stored on the current **HttpContext**'s **User** object.

To get a reference to the current **RestIdentity** of the current **HttpContext**, cast **HttpContext.Current.User.Identity** to **Lullaby.Security.RestIdentity**.

```
var restIdentity =
    (RestIdentity)HttpContext.Current.User.Identity;
```

This will provide access to the **Identity** property as well as the security token that was issued on authentication. The security token stores the following information:

- **ImpersonateUserName** – Username a user was impersonated as.
- **Roles** – Roles that a user is assigned to.
- **TokenId** – The unique identifier of the security token.
- **UserName** – Username that was used to authenticate a user.
- **ValidFrom** – UTC date and time that a security token is valid from.
- **ValidTo** – UTC date and time that a security token is valid to.

If authentication fails, the request will stop from being processed and a 401 status code and **Unauthorized** status description will be set.

Caching

Lullaby has two built-in caching providers: `MemoryCacheProvider` and `FileCacheProvider`. To create a cache provider, create a class that implements `ICacheProvider`.

In order for caching to work properly, cache invalidation needs to be implemented using the `CacheInvalidationMatch` attribute and/or the cache invalidation XML file.

Below is an example of using the `CacheInvalidationMatch` attribute.

```
[CacheInvalidationMatch("/Address/{0}", "addressID"),
CacheInvalidationMatch("/Address/all"),
CacheInvalidationMatch("/Address/all\\?limit=\\d", UseRegularExpressions
= true)]
[RestMethod("{addressID}", HttpMethod.Method.DELETE)]
public void DeleteAddress(int addressID) { ... }
```

Below is the schema of the cache invalidation XML file.

```
<?xml version="1.0" encoding="utf-8"?>
<CacheInvalidationSchema xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Classes>
    <Class>
      <Type />
      <Methods>
        <Method>
          <Name />
          <ReturnType />
          <Parameters>
            <Parameter>
              <Name />
              <Type />
            </Parameter>
          </Parameters>
          <CacheInvalidationMatches>
            <CacheInvalidationMatch>
              <ResourceTemplate />
              <ParameterNames />
              <UseRegularExpressions />
            </CacheInvalidationMatch>
          </CacheInvalidationMatches>
        </Method>
      </Methods>
    </Class>
  </Classes>
</CacheInvalidationSchema>
```

Note: The return type of a void method should be `System.Void`. Class and method names, types, and parameters need to match that of the class and method. Lullaby matches the contents of the cache invalidation file to REST classes and methods.

The first time the cache invalidation file is loaded, it is cached in memory. If a new version of the cache invalidation file is uploaded and it doesn't reside in the websites bin directory, web application will need to be restarted for changes to take effect.

Serialization

Lullaby has two built-in serializers: `JsonSerializer` and `XmlSerializer`. To create a serializer, implement `ISerializer`.

The `SerializationTable` contains mappings of MIME types to serializers. Using `SerializationTable`, MIME type mappings can be added and removed. For example, if a serializer named `MySerializer` was added, the following code would map it for serialization and deserialization:

```
SerializationTable.Serializers.Add("application/mymimetype", new  
MySerializer());
```

The following MIME types are handled by Lullaby by default:

- `application/json`
- `application/x-www-form-urlencoded`
- `application/javascript`
- `application/json+javascript`
- `application/xml`
- `text/javascript`

Client Library

Lullaby includes a client library that allows for REST services to be easily consumed by a .NET client. The client library `RestClient` has the following methods:

- `ConvertResponseToType<T>(WebRequest)` - `ConvertResponseToType` takes a `WebRequest` and converts the response stream to the specified type based on the content type of the request.
- `ConvertResponseToType<T>(string, WebRequest)` - `ConvertResponseToType` takes a `WebRequest` and converts the response stream to the specified type based on the content type provided.
- `GetResponse(WebRequest)` - `GetResponse` takes a `WebRequest` and returns the response stream as a string.
- `GetResponse(WebResponse)` - `GetResponse` takes a `WebResponse` and returns the response stream as a string.