ERRest Framework

Presentations

- A presentation made by Mike Schrag on February 16th 2010 about ERREST. http://wocommunity.org/podcasts/ERRest-2010-02-16.mov It's also listed on the Screencasts Page on wocommunity.org, in iTunes and in the Podcasts RSS Feed.
- Mike also did a session about it at WOWODC West 2009 (see below).
- A session on ERRest integration with Dojo was made by Pascal Robert at WOWODC 2010. Two other sessions about ERRest were made at WOWODC 2011.
- You can also find the slides from Pascal's three WOWODC 2012 about ERRest on SlideShare.

Current and past WOWODC Sessions are available for purchase on the WebObjects Community Association website.

1. Log in, or create an account here: http://www.wocommunity.org/account
2. Go to the Community Store where there will be options for purchasing memberships

Class names vs Entity names

Everything internally in ERRest is based on entity names, as it is throughout WebObjects and EOF.

Most times your Entity Name will be the same as your Class name, but because Entity names must be unique there are situations where you will need to change the Entity name i.e., if you have two Classes named the same thing but are in different packages.

Lets say you have a class named "School" in your model, but have defined the Entity name as "ERPSchool" you will always refer to it as "ERPSchool" when passing it in as a parameter. For example:

```
routeRequestHandler.addDefaultRoutes("ERPSchool"); // School.ENTITY_NAME
```

Also, when naming your controller classes, you should use the Entity name.

```
public class ERPSchoolController extends ERXDefaultRouteController {
...
  School school = (School) routeObjectForKey("ERPSchool");
```

If you want to call it "School" to the outside, add this before you register the default routes for "ERPSchool":

```
ERXRestNameRegistry.registry().setExternalNameForInternalName("School",
"ERPSchool"); // "School", School.ENTITY_NAME
```

After adding this, no other code changes. All you're saying is that the routes and type names that send over the wire should all say "School".

WO HTTP adaptor

The WO HTTP adaptor coming with WO 5.3 doesn't support PUT and DELETE operations, so you won't be able to use those two HTTP methods. You need to use either the 5.4 adaptor or the Wonder version of the adaptor.

To-many relationships

ERRest won't let you add or remove objects in a to-many relationship, it can only update an existing object that is in the relationship. To add a new object to a relationship, you first need to fetch (or create) the parent object and make a second call to create the object and add it to the relationship.

So let's say you have an existing Organization object and you want to add a Member to it. First, you need to fetch the Organization :

```
GET /cgi-bin/WebObjects/ra/Organization/1.json
```

and after, you create a new Member :

```
POST /cgi-bin/WebObjects/ra/Organization/1/addMember.json
```

Same Origin policy

If you are planning to offer your REST services to other people, they might run into Same Origin Policy problem. When using XMLHttpRequest on a page who is on a different domain than the REST service, XMLHttpRequest will tell you that it's not acceptable.

To get around that problem, many solutions has been found, but two of them are more accepted than the others : JSONP and HTTP_access_control. I didn't try the JSONP route, but I did try HTTP_access_control, and Dojo and Prototype are using this method to get around the Same Origin policy problem.

HTTP_access_control works by adding new headers in the request that says which HTTP methods the request wants to do, and it's send as a OPTIONS HTTP method. You MUST reply with some headers to this OPTIONS request. Support for those headers was added in ERRest somewhere in october 2010.

To enable the headers and support for the OPTIONS method, add this property:

```
ERXRest.accessControlAllowOrigin=*
```

If you want to allow origin for only a specific host, change the value of the property to the IP or DNS name of the requester.

Now, sadly only newer (2008 or later) browsers support the HTTP Access Control standard. For older browsers, you have to support JSONP, or the window.name transport. To enable window.name transport support, do it with the following property.

```
ERXRest.allowWindowNameCrossDomainTransport=true
```

Lastly, you can now use JSONP. JSONP works by requesting a callback method name in the URL (?callback=MyCallbackMethodName) and the response is wrapper inside a script tag. Again, you need to enable it:

```
ERXRest.allowJSONP=true
```

Dates

The default formatter for dates is :

```
%Y-%m-%dT%H:%M:%SZ
```

If you want to work with the the GMT offset, you have to use this instead :

```
%Y-%m-%dT%H:%M:%S%z
```

To change it, you have to set the "er.rest.timestampFormat" property :

```
er.rest.timestampFormat = %Y-%m-%dT%H:%M:%S%z
```

If you are using Dojo, you can use dojo.date.stamp.toISOString and dojo.date.stamp.fromISOString to convert from or to a Java date object.

JSON Schema support

ERRest have support for JSON Schema since october 2010. JSON Schema is like a mini-WSDL, where it describe which properties JSON objects have. Dojo use JSON Schema to perform client-side validation, which is cool because you can have some automatic validation based on your model, and validation is done even before the data is sent back to your REST service.

To get the schema, you need to check in your route action. For example, for the "index" action, you do:

```
public WOActionResults indexAction() throws Throwable {
    if (isSchemaRequest()) {
      return schemaResponse(yourERXKeyFilter());
    }
... do your normal work here
  }
```

And you call your REST route, but by adding "?schema=true" at the end of the URL. For example, */ra/events.json?schema=true*. When you do that, a JSON Schema will be returned. Sample:

```
{
  "name":"Event",
  "properties":{
    "isFullDay":{
      "optional":true,
      "type":"string",
      "maxLength":5
    },
    "startDate":{
      "optional":false,
      "type":"string",
      "format":"date-time"
    },
    "realDuration":{
      "optional":true,
      "type":"integer"
    }
  }
}
```

So when you create a new Event with Dojo, if you try to put something else than a date in startDate, or you put a string in realDuration, it will fail because the schema says that startDate is using the date-time format, and that realDuration is a integer.

Getting the URI of a REST route

For whatever reason, you might need to know the URI for a REST route. To get that information, use the ERXRouteController._controllersForRequest(WORequest) method, and get the value of request().uri(). Sample:

```
NSMutableArray<ERXRouteController> routeControllers =
ERXRouteController._controllersForRequest(this.context().request());
if (routeControllers.count() > 0) {
    return routeControllers.objectAtIndex(0).request().uri()
} else {
    return this.context().request().uri()
}
```

Automatic HTML routing

Not only you can get XML and JSON out of ERRest, but you can also get rendered components. This give you "clean" URLs without having to write a bunch of Direct Actions.

To enable the automatic HTML routing, you need to override the isAutomaticHtmlRoutingEnabled method in your controller to return true:

```
public class EventsController extends ERXDefaultRouteController {
...
  protected boolean isAutomaticHtmlRoutingEnabled() {
    return true;
  }
```

When do, you can now add .html when you reach a REST route with a browser, for example /ra/events.html instead of /ra/events.json. But you will notice that you will get an exception because it cannot find a component. You need to create a component with the following pattern:

EntityActionPage

For example, if you have a Event entity that is controlled by EventsController, and you call the "index" REST route to get a list of events, the component name have to be:

EventIndexPage

Also, for all actions except "index", your component have to implement IERXRouteComponent, and you have to add setter methods to get the data that the REST route is sending. For example, if you create a EventShowPage to show the details of a event, EventShowPage.java need the following method:

```
@ERXRouteParameter
   public void setEvent(Event event) {
         ...
   }
```

The @ERXRouteParameter annotation will make the method suitable for "accepting" objects from the controller, if the annotation is not there, you won't be able to receive the object from the controller.

Using D2W rules with ERRest

This is a tip that was sent by Farrukh Ijaz in the webobjects-dev mailing list. It allow you to use D2W rules to create your filters on your REST routes.

```
private D2WContext d2wContext;

protected D2WContext d2wContext() {
 if (d2wContext == null) {
  d2wContext = new D2WContext();
 }
 return d2wContext;
}

synchronized public NSArray<String> inferFilterKeysForEntity(String
entityName) {
 EOEntity entity = EOModelGroup.defaultGroup().entityNamed(entityName);
 d2wContext().setEntity(entity);
 d2wContext().setTask(request().method());
 return (NSArray<String>)
d2wContext().inferValueForKey("displayPropertyKeys");
}

synchronized public NSArray<String> inferFilterKeysForPage(String page) {
 d2wContext().setDynamicPage(page);
 d2wContext().setTask(request().method());
 return (NSArray<String>)
d2wContext().inferValueForKey("displayPropertyKeys");
}

public ERXKeyFilter showFilter(String entityName) {
 NSArray<String> keys = inferFilterKeysForEntity(entityName);
 ERXKeyFilter filter = null;
 if (keys != null && !keys.isEmpty()) {
  filter = ERXKeyFilter.filterWithNone();
  for (String key : keys) {
   filter.include(new ERXKey<String>(key));
  }
 } else {
  filter = ERXKeyFilter.filterWithAttributesAndToOneRelationships();
 }
 return filter;
}
```

The D2WRule should be something like:

```
LHS: entity.name = 'Employee' and task = 'GET'
RHS: displayPropertyKeys = ('firstName', 'lastName', 'position')
```