

Documentum REST Extensibility Tutorial (8): Create A Root Resource

Version 2

created by William Zhou on Aug 3, 2016 2:23 AM, last modified by William Zhou on Aug 3, 2016 7:57 AM



Previous: [Documentum REST Extensibility Tutorial \(7\): Make Resource Batch-able](#)

Preliminary

Before diving into this tutorial, you must at least complete the tutorial [Documentum REST Extensibility Tutorial \(2\): Create A Sample Project](#).

Overview

In this tutorial, we will implement a new resource to list all docbrokers. It is taken as a root resource because it is a topmost resource that does not require an authentication. As you know, Documentum REST Services' navigation starts from a *Home Document*. We will add the root resource *Docbrokers Resource* into Home Document with a link relation <http://identifiers.emc.com/linkrel/brokers>.

Resource	URI	HTTP Method	Mime Type
Docbrokers	/brokers	GET	application/xml
			application/vnd.emc.documentum+xml

In this tutorial, you will learn the complete lifecycle of implementing a standalone resource, from **model**, **persistence**, **controller**, view to **YAML configuration**.

Please note, as a demonstration, we decide to support XML media types only in Docbrokers Resource.

Implementation

Here is the sheet of files we are going to modify in the project created by [Documentum REST Extensibility Tutorial \(2\): Create A Sample Project](#). The project location is at `<SDK_ROOT>/maven-kit/generated/acme-rest/`. The file name with asterisk (*) means the file is newly added.

- `acme-rest-model/src/main/java/com/acme/rest/model/Docbroker.java*`
- `acme-rest-model/src/main/java/com/acme/rest/model/DocbrokerList.java*`
- `acme-rest-persistence/src/main/java/com/acme/rest/dfc/DocbrokerManager.java*`
- `acme-rest-resource/src/main/java/com/acme/rest/controller/DocbrokersController.java*`
- `acme-rest-resource/src/main/java/com/acme/rest/view/impl/DocbrokerListView.java*`
- `acme-rest-web/src/main/resources/com/emc/documentum/rest/script/rest-api-custom-resource-registry.yaml`

Model

We need to design the resource representation for the docbroker list. Since it only supports XML media types, we want the representation to be like this:

```

01. <docbrokers>
02.   <count>1</count>
03.   <protocol>rpc_static</protocol>
04.   <rpc-mode>1</rpc-mode>
05.   <brokers>
06.     <b>
07.       <host>192.168.1.100</host>
08.       <port>1489</port>
09.       <timeout>0</timeout>
10.     </b>
11.     <b>
12.       <host>192.168.1.101</host>
13.       <port>1490</port>
14.       <timeout>90</timeout>
15.     </b>
16.   </brokers>
17.   <links>
18.     <link rel="self" href="http://localhost:8080/acme-rest/brokers"/>
19.   </links>
20. </docbrokers>

```

Here we use Documentum REST annotations *@SerializableType* and *@SerializableField* to create the resource model classes in module *acme-rest-model*.

Create *DocbrokerList.java* as the container of docbroker list.

```

01. @SerializableType("docbrokers")
02. public class DocbrokerList extends AbstractLinkable {
03.     private int count;
04.     private String protocol;
05.     @SerializableField("rpc-mode")
06.     private int mode;
07.     @SerializableField(xmlListItemName = "b")
08.     private List<Docbroker> brokers;
09.
10.     public void setProtocol(String protocol) {
11.         this.protocol = protocol;
12.     }
13.     public void setMode(int mode) {
14.         this.mode = mode;
15.     }
16.     public void setCount(int count) {
17.         this.count = count;
18.     }
19.     public List<Docbroker> getBrokers() {
20.         if (brokers == null) {
21.             brokers = new ArrayList<Docbroker>();
22.         }
23.         return brokers;
24.     }
25. }

```

Create *Docbroker.java* as the item of a single docbroker.

```

01. @SerializableType("docbroker")
02. public class Docbroker {
03.     private String host;
04.     private int port;
05.     private int timeout;

```

```

06.
07.     public void setHost(String host) {
08.         this.host = host;
09.     }
10.     public void setPort(int port) {
11.         this.port = port;
12.     }
13.     public void setTimeout(int timeout) {
14.         this.timeout = timeout;
15.     }
16. }

```

Persistence

By DFC, we need to call DFC local client to get the docbroker map. The docbroker information comes from *dfc.properties* in the REST WAR file.

In module *acme-rest-persistence*, we create the class *DocborkerManager.java*.

```

01.     public class DocbrokerManager {
02.         public DocbrokerList listAll() throws DfException {
03.             DocbrokerList brokers = new DocbrokerList();
04.             IDfTypedObject docbrokerMap = new DfClientX().getLocalClient().getDocbrokerMap();
05.             int count = docbrokerMap.getValueCount("host_name");
06.             brokers.setCount(count);
07.             brokers.setProtocol(docbrokerMap.getString("network_protocol"));
08.             brokers.setMode(docbrokerMap.getInt("secure_connect_mode"));
09.             for (int k=0; k<docbrokerMap.getValueCount("host_name"); k++) {
10.                 Docbroker broker = new Docbroker();
11.                 broker.setHost(docbrokerMap.getRepeatingString("host_name", k));
12.                 broker.setPort(docbrokerMap.getRepeatingInt("port_number", k));
13.                 broker.setTimeout(docbrokerMap.getRepeatingInt("time_out", k));
14.                 brokers.getBrokers().add(broker);
15.             }
16.             return brokers;
17.         }
18.     }

```

Controller

Now in module *acme-rest-resource*, let's define the resource controller. Here is the class *DocbrokersController.java*.

```

01.     @Controller("acme#docbrokers")
02.     @RequestMapping("/brokers")
03.     @ResourceViewBinding(value = DocbrokerListView.class)
04.     public class DocbrokersController extends AbstractController {
05.         @RequestMapping(
06.             method = RequestMethod.GET,
07.             produces = {
08.                 SupportedMediaTypes.APPLICATION_VND_DCTM_XML_STRING,
09.                 MediaType.APPLICATION_XML_VALUE
10.             }
11.         )
12.         @ResponseBody
13.         @ResponseStatus(HttpStatus.OK)
14.         public DocbrokerList getDocbrokers(
15.             @RequestUri final UriInfo uriInfo)
16.             throws Exception {

```

```

17. DocbrokerList brokers = new DocbrokerManager().listAll();
18. return getRenderedObject(null, brokers, true, uriInfo, null);
19. }
20. }

```

The resource is designed as *GET* on URI */brokers*. It supports XML media types.

View

You have noticed that the resource controller is bound to a view class *DocbrokerListView.java*. Here we create this class in module *acme-rest-resource*. The only purpose of the view implementation is to build the *self* link for *Docbrokers Resource*. Here is the code.

```

01. @DataViewBinding(modelType = DocbrokerList.class)
02. public class DocbrokerListView extends LinkableView<DocbrokerList> {
03.     public DocbrokerListView(DocbrokerList serializableData, UriInfo uriInfo,
04.         String repositoryName, Boolean returnLinks, Map<String, Object> others) {
05.         super(serializableData, uriInfo, repositoryName, returnLinks, others);
06.     }
07.     @Override protected Map<String, Object> resolveUriTemplateVariables(Map<String, String> map) {
08.         return Collections.emptyMap();
09.     }
10.     @Override public void customize() {
11.         //do nothing
12.     }
13.     @Override public String canonicalResourceUri(boolean validate) {
14.         return getUriFactory(validate).buildUriByTemplateName("X_DOCBROKERS_URI_TEMPLATE", null);
15.     }
16. }

```

By this step, we have completed all the Java coding. The last step is to configure the YAML file.

YAML Configuration

The YAML file *rest-api-custom-resource-registry.yaml* is in module *acme-rest-web*. We modify the file for two purposes:

1. In the view class *DocbrokerListView.java*, we note a URI template name *X_DOCBROKERS_URI_TEMPLATE*. We define the URI template in the YAML file.
2. Although we create the *Docbrokers Resource* implementation, the client does not know its resource URI ahead and we need to find a way to build the link relation for this resource. The solution is to add this resource to *Home Document*, by the YAML configuration. Please read [Make your Documentum REST clients hypermedia controlled](#) if you want to understand why we need to create link relations for new resources.

Here is the YAML file.

```

01. ---
02. uri-template-registry:
03.   - name: X_ALIAS_SETS_URI_TEMPLATE
04.     href: '{repositoryUri}/alias-sets{ext}'
05.   - name: X_ALIAS_SET_URI_TEMPLATE
06.     href: '{repositoryUri}/alias-sets/{aliasSetId}{ext}'
07. # add start
08.   - name: X_DOCBROKERS_URI_TEMPLATE
09.     href: '{baseUri}/brokers'
10. # add end

```

```

11.
12.
13. # add start
14. ---
15. root-service-registry:
16.   - name: 'docbrokers reosurce'
17.     link-relation: 'http://identifiers.emc.com/linkrel/brokers'
18.     uri-template: X_DOCBROKERS_URI_TEMPLATE
19.     allowed-methods: ['GET']
20.     media-types: ['application/vnd.emc.documentum+xml', 'application/xml']
21. # add end
22.
23.
24. ---
25. resource-link-registry:
26.   - resource: repository
27.     link-relation: 'http://identifiers.emc.com/linkrel/alias'
28.     uri-template: X_ALIAS_SETS_URI_TEMPLATE
29.     value-mapping: []

```

Deploy and Test

Run Maven goal *mvn clean install* to rebuild the project and redeploy it to the web container again.

Open a REST client debugging tool like PostMan or DHC to test the result.

Request - Get home document:

Get *Home Document* by URI */services*. This is the only URI that the REST client can hard-code.

```

01. GET http://localhost:8080/acme-rest/services HTTP/1.1
02. Accept: */*

```

Response - Get home document:

From *Home Document*, you will find a new link relation *http://identifiers.emc.com/linkrel/brokers* for *Docbrokers Resource*. Please note the default media type in response is JSON.

```

01. HTTP/1.1 200 OK
02. Content-Type: application/vnd.emc.documentum+json;charset=UTF-8
03.
04.
05. {
06.   "resources":{
07.     "http://identifiers.emc.com/linkrel/repositories":{
08.       "href":"http://localhost:8080/acme-rest/repositories",
09.       "hints":{"allow":["GET"]},
10.       "representations":["application/xml","application/json","application/atom+xml","application/vnd.
11.     },
12.     "about":{
13.       "href":"http://localhost:8080/acme-rest/product-info",
14.       "hints":{"allow":["GET"]},
15.       "representations":["application/xml","application/json","application/vnd.emc.documentum+xml","ap
16.     },
17.     "http://identifiers.emc.com/linkrel/brokers":{
18.       "href":"http://localhost:8080/acme-rest/brokers",
19.       "hints":{"allow":["GET"]},
20.       "representations":["application/vnd.emc.documentum+xml","application/xml"]}

```

```
21.     }  
22.   }  
23.  
24. }
```

Request - Get docbrokers:

Get *Docbrokers Resource* follow the href of the link relation <http://identifiers.emc.com/linkrel/brokers>.

```
01. GET http://localhost:8080/acme-rest/brokers HTTP/1.1  
02. Accept: */*
```

Response - Get docbrokers:

Please note the response media type is XML.

```
01. <docbrokers>  
02.   <count>1</count>  
03.   <protocol>rpc_static</protocol>  
04.   <rpc-mode>1</rpc-mode>  
05.   <brokers>  
06.     <b>  
07.       <host>192.168.1.100</host>  
08.       <port>1489</port>  
09.       <timeout>0</timeout>  
10.     </b>  
11.   </brokers>  
12.   <links>  
13.     <link rel="self" href="http://localhost:8080/acme-rest/brokers"/>  
14.   </links>  
15. </docbrokers>
```

By this tutorial, you have learnt the complete lifecycle of implementing a new resource. You've also learnt how to configure a root resource. Well done!

Next: [Documentum REST Extensibility Tutorial \(9\): Create Persistence Managers](#)