

# Documentum REST Extensibility Tutorial (9): Create Persistence Managers

Version 3

created by William Zhou on Aug 3, 2016 2:24 AM, last modified by William Zhou on Nov 20, 2016 7:37 PM



**Previous:** [Documentum REST Extensibility Tutorial \(8\): Create A Root Resource](#)

## 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 show a complete sample of implementing a tree view resource on folders and cabinets. The purpose of this tutorial is to tell how to write DFC code in REST implementation, but you will also learn the complete development for a resource.

We know that a cabinet or folder can have descendants of sub folders or documents. Documentum REST Services by default provides you resources to get folder children and folder parents, but it does not have the resource for folder tree since a tree representation in a single call can be super huge.

In this tutorial, we implement the *Folder Tree Resource* with very simple representation. We also use a query parameter *depth* to control how many levels of descendants we want to get from the tree root.

- For a sub folder, we return the folder name and its descendants
- For a sub document, we return the document name, size, and format.
- No link relations for the folder tree nodes.

Here is the resource design.

Resource	URI	HTTP Method	Mime Type
Folder Tree	/repositories/{repositoryName}/folders/{folderId}/tree	GET	application/xml application/vnd.emc.documentum application/json application/vnd.emc.documentum

From a *Cabinet Resource* or a *Folder Resource*, we add a link relation '<http://identifiers.emc.com/linkrel/tree>' to navigate to *Folder Tree Resource*.

On *Folder Tree Resource*, we also add a link relation '[up](#)' navigating back to the parent cabinet or folder.

## 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/FolderTree.java\**
- *acme-rest-model/src/main/java/com/acme/rest/model/Node.java\**
- *acme-rest-model/src/main/java/com/acme/rest/model/FolderNode.java\**
- *acme-rest-model/src/main/java/com/acme/rest/model/DocNode.java\**
- *acme-rest-persistence/src/main/java/com/acme/rest/dfc/FolderManager.java\**
- *acme-rest-persistence/src/main/java/com/acme/rest/dfc/FolderManagerImpl.java\**
- *acme-rest-persistence/src/main/resources/META-INF/spring/rest-api-acme-persistence.xml\**
- *acme-rest-resource/src/main/java/com/acme/rest/controller/FolderTreeController.java\**
- *acme-rest-resource/src/main/java/com/acme/rest/view/impl/FolderTreeView.java\**
- *acme-rest-web/src/main/resources/com/emc/documentum/rest/script/rest-api-custom-resource-registry.yaml*

## Model

In module *acme-rest-model*, we design the folder tree representation. Here are the complete code of model classes.

### Folder Tree

```

01. @SerializableType(value = "tree", fieldVisibility = SerializableType.FieldVisibility.NONE, fieldOrder :
02. public class FolderTree extends AbstractLinkable {
03.     private String id;
04.     @SerializableField
05.     private String name;
06.     @SerializableField(value = "children", xmlListItemName = "child")
07.     private List<Node> nodes;
08.
09.
10.     public void setId(String id) {
11.         this.id = id;
12.     }
13.     public void setName(String name) {
14.         this.name = name;
15.     }
16.     public List<Node> getNodes() {
17.         if (nodes == null) {
18.             nodes = new ArrayList<Node>();
19.         }
20.         return nodes;
21.     }
22.     public String getId() {
23.         return id;
24.     }
25. }

```

### Abstract Tree Node

```

01. @SerializableType("node")
02. public abstract class Node {
03.     protected String name;
04. }

```

### Folder Tree Node

```

01. @SerializableView(value = "folder", fieldOrder = {"name", "type", "children"})
02. public class FolderNode extends Node {
03.     private String type = "folder";
04.     @SerializableField(xmlListitemName = "child")
05.     private List<Node> children;
06.
07.
08.     public FolderNode() {}
09.     public FolderNode(String name, Node... children) {
10.         this.name = name;
11.         if (children != null) {
12.             for (Node child : children) {
13.                 getChildren().add(child);
14.             }
15.         }
16.     }
17.     public List<Node> getChildren() {
18.         if (children == null) {
19.             children = new ArrayList<Node>();
20.         }
21.         return children;
22.     }
23. }

```

### Document Tree node

```

01. @SerializableView(value = "doc", fieldOrder = {"name", "type", "format", "size"})
02. public class DocNode extends Node {
03.     private String type = "document";
04.     private String format;
05.     private long size;
06.
07.
08.     public DocNode() {}
09.     public DocNode(String name, String format, long size) {
10.         this.name = name;
11.         this.format = format;
12.         this.size = size;
13.     }
14. }

```

## Persistence

In module *acme-rest-persistence*, we write DFC code to implement the service. We first create a *FolderManager* interface.

```

01. public interface FolderManager {
02.     FolderTree getFolderTree(String folderId, int depth) throws DfException;
03. }

```

The implementation class will extend from Documentum REST library *SessionAwareAbstractManager* to get DFC sessions. For a folder or cabinet, its *dm\_folder* and sub type descendants, and *dm\_document* and sub type descendants are returned by depth.

```

01. public class FolderManagerImpl extends SessionAwareAbstractManager implements FolderManager {
02.     private static final String ATTRS = "r_object_id,object_name,a_content_type,r_object_type,r_full_c
03.

```

```

04.
05. @Override public FolderTree getFolderTree(String folderId, int depth) throws DfException {
06.     IDfSession session = null;
07.     try {
08.         session = getSessionRepository().getSession();
09.         IDfFolder folder = (IDfFolder) session.getObject(new DfId(folderId));
10.         FolderTree tree = new FolderTree();
11.         tree.setId(folderId);
12.         tree.setName(folder.getObjectName());
13.         if (depth != 0) {
14.             tree.getNodes().addAll(getFolderChildren(folder, session, depth));
15.         }
16.         return tree;
17.     }
18.     finally {
19.         DfcSessions.release(session);
20.     }
21. }
22. private List<Node> getFolderChildren(IDfFolder folder, IDfSession session, int depth) throws DfExc
23.     if (depth == 0) {
24.         return Collections.emptyList();
25.     }
26.
27.
28.     IDfCollection children = folder.getContents(ATTRS);
29.     List<Node> result = new ArrayList<Node>();
30.     while (children.next()) {
31.         IDfTypedObject child = children.getTypedObject();
32.         IDfType objectType = session.getType(child.getString("r_object_type"));
33.         if (isTypeOf(objectType, "dm_folder", session)) {
34.             result.add(createFolderNode(session, depth, child));
35.         }
36.         if (isTypeOf(objectType, "dm_document", session)) {
37.             result.add(createDocNode(child));
38.         }
39.     }
40.     return result;
41. }
42. private FolderNode createFolderNode(IDfSession session, int depth, IDfTypedObject child) throws Df
43.     FolderNode node = new FolderNode(child.getString("object_name"));
44.     IDfFolder subFolder = (IDfFolder) session.getObject(child.getId("r_object_id"));
45.     List<Node> subChildren = getFolderChildren(subFolder, session, depth - 1);
46.     node.getChildren().addAll(subChildren);
47.     return node;
48. }
49. private DocNode createDocNode(IDfTypedObject child) throws DfException {
50.     return new DocNode(
51.         child.getString("object_name"),
52.         child.getString("a_content_type"),
53.         child.getLong("r_full_content_size")
54.     );
55. }
56. private boolean isTypeOf(IDfType typeToCheck, String baseType, IDfSession session) throws DfExcept
57.     if (typeToCheck == null) {
58.         return false;
59.     }
60.     return typeToCheck.isTypeOf(baseType) ||
61.         typeToCheck.isSubTypeOf(baseType) ||
62.         isTypeOf(typeToCheck.getSuperType(), baseType, session);
63. }
64. }

```

Please see how session is obtained in **line 08**.

This FolderManager shall be called by the controller class. So we declare the initialization of the FolderManager in Spring bean configuration file *rest-api-acme-persistence.xml*.

```
01. <beans xmlns="http://www.springframework.org/schema/beans"
02.       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
03.       xsi:schemaLocation="http://www.springframework.org/schema/beans
04.       http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">
05.     <bean id="folderManager" class="com.acme.rest.dfc.FolderManagerImpl"/>
06. </beans>
```

## Controller

In module *acme-rest-resource*, we create the resource controller class *FolderreeController.java*.

```
01. @Controller("acme#folder-tree")
02. @RequestMapping("/repositories/{repositoryName}/folders/{folderId}/tree")
03. @ResourceViewBinding(value = FolderTreeView.class)
04. public class FolderTreeController extends AbstractController {
05.     @Autowired
06.     FolderManager folderManager;
07.
08.
09.     @RequestMapping(
10.         method = RequestMethod.GET,
11.         produces = {
12.             SupportedMediaTypes.APPLICATION_VND_DCTM_JSON_STRING,
13.             SupportedMediaTypes.APPLICATION_VND_DCTM_XML_STRING,
14.             MediaType.APPLICATION_JSON_VALUE,
15.             MediaType.APPLICATION_XML_VALUE
16.         }
17.     )
18.     @ResponseBody
19.     @ResponseStatus(HttpStatus.OK)
20.     public FolderTree getFolderTreeView(
21.         @PathVariable("repositoryName") final String repositoryName,
22.         @PathVariable("folderId") final String folderId,
23.         @RequestParam(value = "depth", defaultValue = "-1") final int depth,
24.         @RequestUri final UriInfo uriInfo)
25.         throws Exception {
26.         FolderTree tree = folderManager.getFolderTree(folderId, depth);
27.         return getRenderedObject(repositoryName, tree, true, uriInfo, null);
28.     }
29. }
```

## View

In the same module *acme-rest-resource*, we create the resource view binding class *FolderTreeView.java*.

```
01. @DataViewBinding(modelType = FolderTree.class)
02. public class FolderTreeView extends LinkableView<FolderTree> {
03.     public FolderTreeView(FolderTree 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.         makeLink("up", getUriFactory().folderUri(getDataInternal().getId(), null));
12.     }
13.     @Override public String canonicalResourceUri(boolean validate) {
14.         return getUriFactory(validate).buildUriByTemplateName(
15.             "X_FOLDER_TREE_URI_TEMPLATE",
16.             Collections.singletonMap("folderId", getDataInternal().getId()));
17.     }
18. }

```

As the code shows, it creates two link relations '*self*' (by parent view class) and '*up*'. Please note here we build the self link by a URI template name *X\_FOLDER\_TREE\_URI\_TEMPLATE*. The URI template will be defined in 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\_FOLDER\_TREE\_URI\_TEMPLATE*. We define the URI template in the YAML file.
2. As mentioned in the Overview, we want to add a link relation '*https://community.emc.com/linkrel/tree*' to *Cabinet Resource* and *Folder Resource*. Please note that the *folderId* variable value in the URI template comes from the attribute value of *r\_object\_id* of a cabinet or folder object.

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_FOLDER_TREE_URI_TEMPLATE
09.     href: '{repositoryUri}/folders/{folderId}/tree{ext}'
10. # add end
11.
12.
13. ---
14. resource-link-registry:
15.   - resource: repository
16.     link-relation: 'http://identifiers.emc.com/linkrel/alias'
17.     uri-template: X_ALIAS_SETS_URI_TEMPLATE
18.     value-mapping: []
19. # add start
20.   - resource: cabinet
21.     link-relation: 'http://identifiers.emc.com/linkrel/tree'
22.     uri-template: X_FOLDER_TREE_URI_TEMPLATE
23.     value-mapping: ['folderId:r_object_id']
24.   - resource: folder
25.     link-relation: 'http://identifiers.emc.com/linkrel/tree'
26.     uri-template: X_FOLDER_TREE_URI_TEMPLATE
27.     value-mapping: ['folderId:r_object_id']

```

```
28. | # add end
```

## 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 a cabinet:*

Navigate to a Cabinet Resource.

```
01. | GET http://localhost:8080/acme-rest/repositories/REPO/cabinets/0c000005800028e7?view=object_name,r_obji
02. | Authorization: Basic ZG1hZG1pbjpwYXNzd29yZA==
03. | Content-Type: */*
```

### *Response - Get a cabinet:*

Please note a link relation <https://community.emc.com/linkrel/tree> in the links collection.

```
01. | HTTP/1.1 200 OK
02. | Content-Type: application/vnd.emc.documentum+json;charset=UTF-8
03. |
04. |
05. | {
06. |   "name":"cabinet",
07. |   "type":"dm_cabinet",
08. |   "definition":"http://localhost:8080/acme-rest/repositories/REPO/types/dm_cabinet",
09. |   "properties":{
10. |     "object_name":"CreatedCabinetName65df8d58-46a3-4925-8d5d-f4e1071337a9",
11. |     "r_object_id": "0c000005800028e7"
12. |   },
13. |   "links":[
14. |     {"rel":"self", "href":"http://localhost:8080/acme-rest/repositories/REPO/cabinets/0c000005800028e7"},
15. |     {"rel":"edit", "href":"http://localhost:8080/acme-rest/repositories/REPO/cabinets/0c000005800028e7"},
16. |     {"rel":"http://identifiers.emc.com/linkrel/tree", "href":"http://localhost:8080/acme-rest/reposito
17. |     ...
18. |   ]
19. | }
```

### *Request - Get folder tree:*

Now follow the link relation <https://community.emc.com/linkrel/tree> to get its folder tree.

```
01. | GET http://localhost:8080/acme-rest/repositories/REPO/folders/0c000005800028e7/tree HTTP/1.1
02. | Authorization: Basic ZG1hZG1pbjpwYXNzd29yZA==
03. | Content-Type: */*
```

### *Response - Get folder tree:*

```
01. | HTTP/1.1 200 OK
02. | Content-Type: application/vnd.emc.documentum+json;charset=UTF-8
03. |
04. |
05. | {
06. |   "name":"CreatedCabinetName65df8d58-46a3-4925-8d5d-f4e1071337a9",
07. |   "children":[
```

```
08.  {
09.    "name": "TargetSetup.txt",
10.    "type": "document",
11.    "format": "crtext",
12.    "size": 3172
13.  },
14.  {
15.    "name": "BPM",
16.    "type": "folder",
17.    "children": [
18.      {
19.        "name": "Method Exec Results",
20.        "type": "folder"
21.      },
22.      {
23.        "name": "Method Exceptions",
24.        "type": "folder"
25.      }
26.    ]
27.  },
28.  {
29.    "name": "FTCreateEvents.pdf",
30.    "type": "document",
31.    "format": "pdf",
32.    "size": 2955
33.  },
34.  {
35.    "name": "RestTestFolder",
36.    "type": "folder"
37.  }
38. ],
39. "links": [
40.   {"rel": "self", "href": "http://localhost:8080/acme-rest/repositories/REPO/folders/0c000005800028e7/1"},
41.   {"rel": "up", "href": "http://localhost:8080/acme-rest/repositories/REPO/folders/0c000005800028e7"}
42. ]
43. }
```

You can also try different *Accept* request headers to test XML and JSON media types. And if you append a query parameter *?depth=1* to the folder tree URI, e.g. <http://localhost:8080/acme-rest/repositories/REPO/folders/0c000005800028e7/tree?depth=1>, you will only get one level of folder tree.

We are done here for this tutorial.

**Next:** [Documentum REST Extensibility Tutorial \(10\): REST Error Handling](#)

[Learn more about Documentum REST Services >>](#)