

# Documentum REST Extensibility Tutorial (3): Explore The Sample Project

---

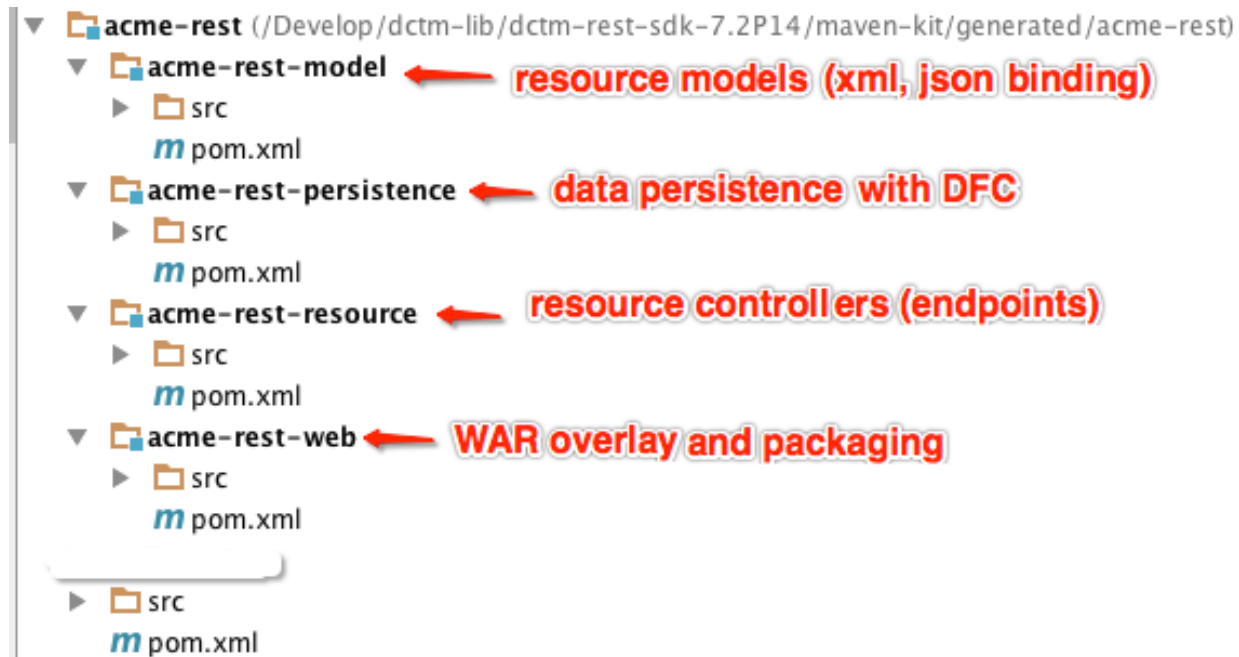
**Previous:** [Documentum REST Extensibility Tutorial \(2\): Create A Sample Project](#)

This document is for the continuous series for Documentum REST Extensibility tutorials. Previously, we talked about how to create a sample project from REST SDK. In this document, we will explore this project to get better understand of the REST extensibility. For the overview of Documentum REST extensibility, please read [Introduction to Documentum REST Extensibility](#).

## Overview

Please open the *acme-rest* project at `<SDK_ROOT>/maven-kit/generated/acme-rest/`. You can use an Java IDE to open the project as a Maven project or open files in the text editor directly.

Here is the code structure of the sample project we have just created. It's a multi-module Maven project.



## acme-rest/pom.xml

- Organize the project structure, dependency, properties and build steps.

## acme-rest/acme-rest-model

- Create Java model classes for resource representations. Model classes are annotated with Documentum REST annotations `@SerializableType` for XML and JSON binding.

## acme-rest/acme-rest-persistence

- Create persistence APIs to interact with DFC. Persistence APIs take the responsibility to save and get persistent objects in Documentum repositories.

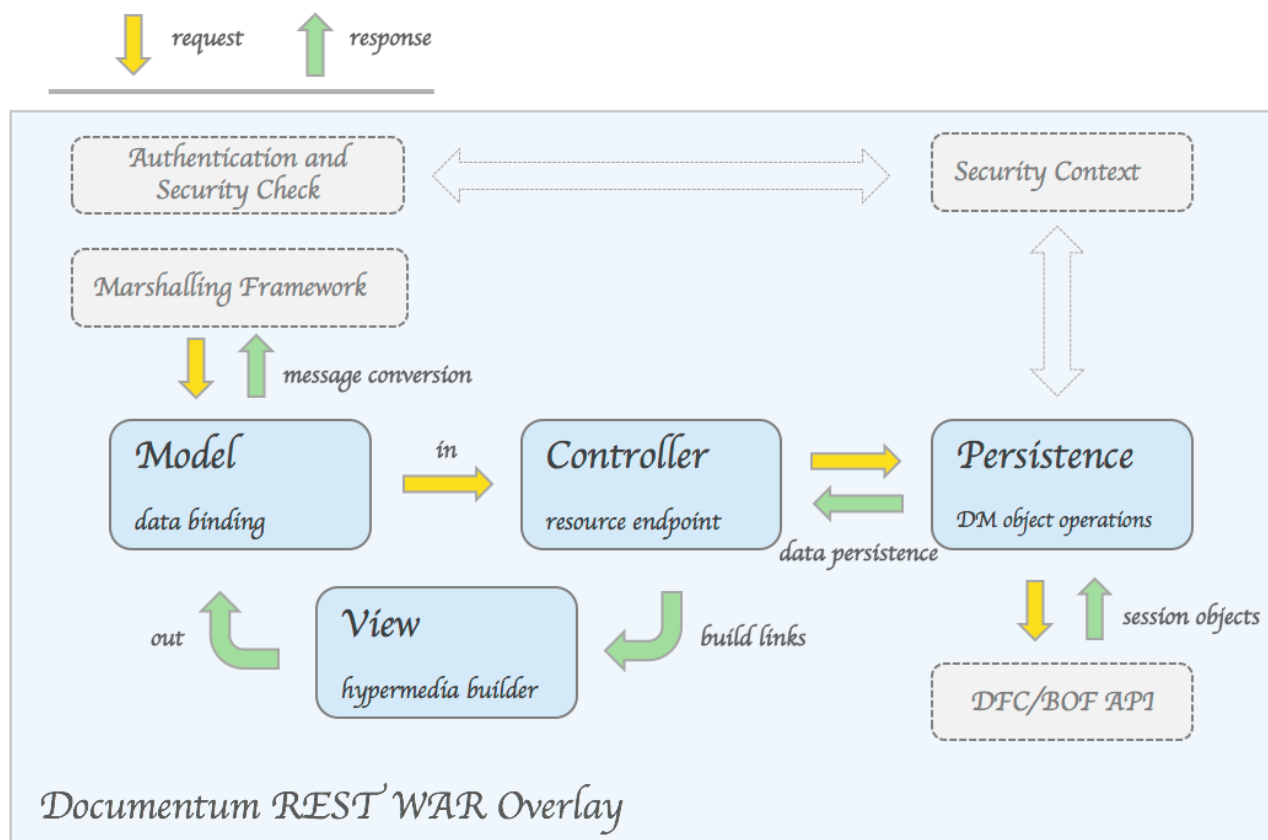
## acme-rest/acme-rest-resource

- Create resource controllers with Spring annotation `@Controller`. Resource controllers define the endpoints for resources, for instance, URIs, request parameters, response bodies and so on.
- Create resource views to customize resource model instances, for instance to build link relations for the resource models.

## acme-rest/acme-rest-web

- Create the WAR overlay (in pom.xml) to customize Documentum Core REST WAR (from your local Maven repository) with your implementations.

Here is a diagram explaining their relations in the REST application.



## Model

The *acme-rest-model* module contains only one class, *AliasSet.java*. It extends from Documentum Core REST class *com.emc.documentum.rest.model.PersistentObject*.

```
@SerializableType(value = "alias-set", xmlNSPrefix = "dm",      xmlINS = "http://identifiers.emc.com/vocab/documentum") public class
```

With the annotation `@SerializableType`, the class is bound to both XML and JSON media types. When it is transferred in the REST API, the XML and JSON messages for an alias set is presented as below.

```
<?xml version='1.0' encoding='UTF-8'?> <alias-set xmlns="http://identifiers.emc.com/vocab/documentum" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://identifiers.emc.com/vocab/documentum http://identifiers.emc.com/vocab/documentum/alias-set.xsd">
  { "type":"dm_alias_set", "definition":"http://localhost:8080/acme-rest/repositories/REPO/types/dm_alias_set", "properties": { "c
```

Please note model classes will be used as the input and out of resource controllers. They are also the data transfer objects between resource controllers and persistence APIs.

For a resource method, you can also different model classes for the input and output.

For more information about designing the REST model with annotations, please refer to *Documentum REST Development Guide - Documentum REST Marshalling Framework*.

## Persistence

The *acme-rest-persistence* module contains only one class *AliasSetCollectionQueryTemplate.java*. This class defines a custom DQL query to get the collection of alias set objects with condition and pagination.

```
public class AliasSetCollectionQueryTemplate extends PagedQueryTemplate {    @Override    protected List<String> defaultFields()
```

Please note in the sample project, we don't create new APIs to call DFC to get the alias objects, e.g. retrieving DFC sessions and releasing DFC sessions. Documentum Core REST persistence library already provides us a lot of persistence APIs that we can use directly. That will be shown in the next section.

## Resource

The *acme-rest-resource* module contains **resource controllers** and **resource views**. As told previously, resource controllers define the endpoint of a REST API with Spring annotation `@Controller`. In this sample project, we have two controller classes:

- `AliasSetCollectionController` (for alias set collection)
- `AliasSetController` (for a single alias set)

Both controller classes have some common settings. Here is the code of *AliasSetController.java*.

```
@Controller("acme#alias-set") @RequestMapping("/repositories/{repositoryName}/alias-sets/{aliasSetId}") @ResourceViewBinding
```

There are some mandatory settings on the class implementation.

- Extend from *com.emc.documentum.rest.controller.AbstractController*
  - By extending from this abstract class, your controller will get the default bindings for request parameter resolvers (from query to Java parameter), the resource view bindings for controller methods (build links for resource models).
- Annotate with *@org.springframework.stereotype.Controller*
  - This Spring annotation makes your controller the implementation of a web endpoint. You should set a custom name for the this annotation.
- Annotate with *@org.springframework.web.bind.annotation.RequestMapping*
  - This Spring annotation defines the request mapping (URI, mime type, params, etc.) for your controller. Any servlet requests that match the request mapping patterns will be dispatched to this controller implementation.
  - RequestMapping can be set on both class level and method level. We recommend that you always define the URI mapping part in the class level RequestMapping.
- Annotate with *@com.emc.documentum.rest.view.annotation.ResourceViewBinding*
  - This is a Documentum REST annotation to bind the resource controller and the resource model to resource views. We will explain a bit later how resource view works.
  - ResourceViewBinding can be set on both class level and method level. For the same output on different controller methods, different views can be bound so that link relations can be customized differently.

The controller class has only one method *getAliasSet()* which returns the alias set resource by object ID. It supports **HTTP GET** method. There are some additional annotations on the method. We let you explore them as a task.

The controller class has an *@Autowired* member *sysObjectManager*. This is Documentum Core REST persistence API bean that is reused by the extension project. When using it, you

don't need to care about session handling, since the session handling is encapsulated in the method implementation.

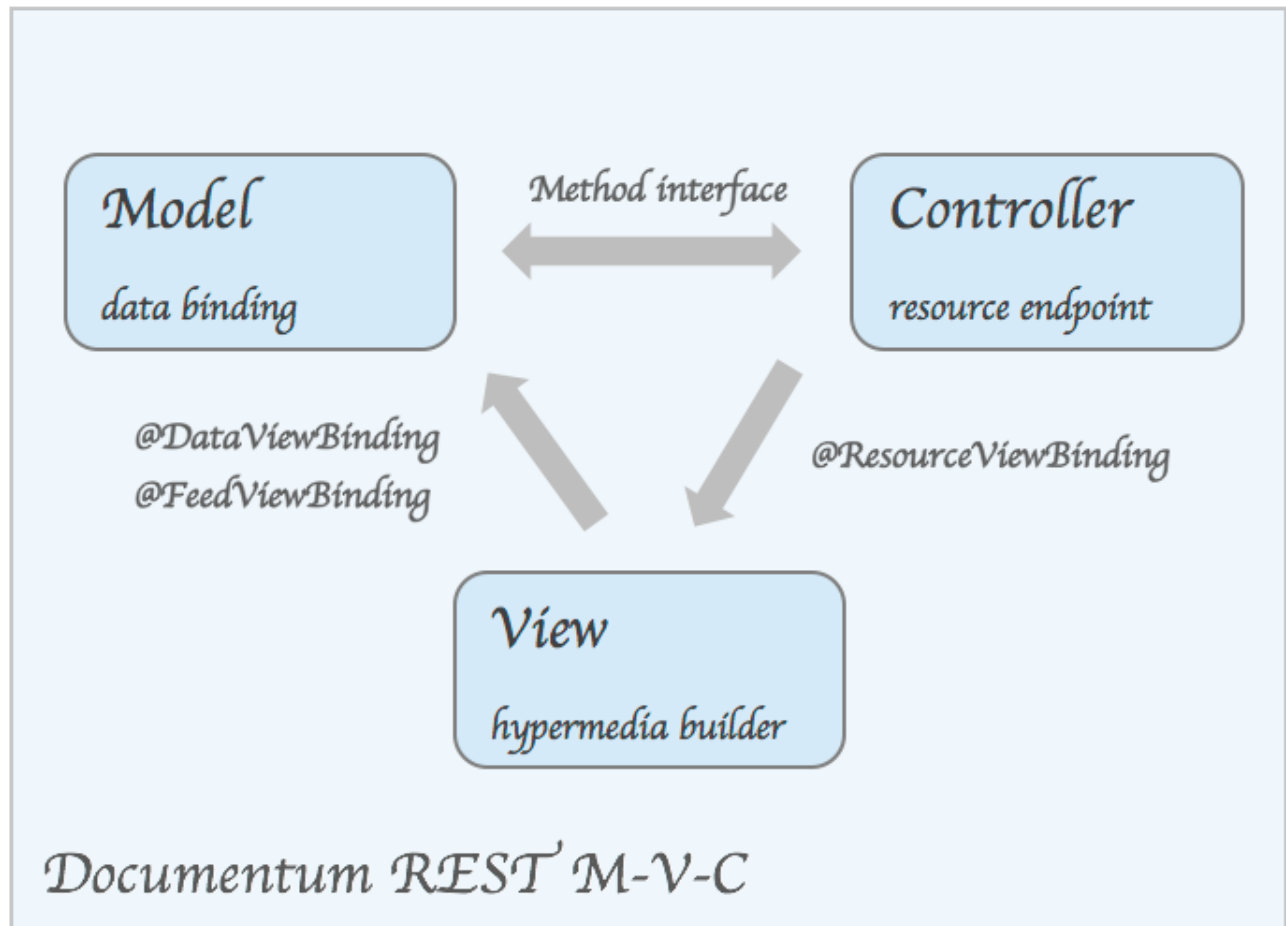
## Spring configuration for controllers

Besides Java classes, we need to create a Spring XML configuration to load the controllers as Spring beans at runtime. The XML file location pattern should be *src/main/resources/META-INF/spring/rest-api-\*.xml*.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?> <beans xmlns="http://www.springframework.org/schema/beans" xm
```

## View

When resource model instances are returned from **Persistence** to **Controller**, they only contain a set of properties. However, a resource representation is hypermedia because it has link relations. Resource views are classes which create link relations for resource models. The relationships among Controller, Model and View can be explained with below diagram.



1. **Resource model** is the input (request body) and output (response body) of **resource controller**.
2. **Resource view** is the hypermedia builder of **resource model**.
3. **Resource controller** is bound to **resource view** with annotation `@ResourceViewBinding`.
4. **Resource view** is bound to **resource model** with annotation `@DataViewBinding` or `@FeedViewBinding`.

In the *acme-rest-resource* module, we have two resource view classes.

- `AliasSetsFeedView`
- `AliasSetView`

Documentum Core REST provides a number of base classes for you to implement the resource views.

- `com.emc.documentum.rest.view.FeedableView<T extends Linkable>`
  - used for views that build links for AtomFeed
- `com.emc.documentum.rest.view.LinkableView<T extends Linkable>`
  - used for views that build links for Linkable
- `com.emc.documentum.rest.view.EntryableView<T extends Linkable>`

- with additional methods to set atom entry attributes over LinkableView
- *com.emc.documentum.rest.view.PersistentLinkableView<T extends PersistentObject>*
  - a default implementation for PersistentObject model

One highlight is, in the class *AliasSetView.java*, there is a method *customize()* to define its own link relation. The below code creates a link relation *author* when the attribute *owner\_name* is not missing on the alias set resource.

```
@Override public void customize() { makeLinkIf(getDataInternal().getAttributeByName("owner_name") != null, LinkRelation.AU
```

## Web

The *acme-rest-web* module mainly does two things:

- Customize REST deployment parameters, including *dfc.properties* and *com/emc/documentm/rest/script/rest-api-custom-resource.yaml*
- Repackage the WAR with both Core REST and extension implementations as the inclusions.

We need to take a detail look at the YAML file. The YAML is the important part of the REST extensibility. It allows users to inject REST extensions (e.g. new links relations) in existing Core REST representation. The YAML configuration below adds a link relation " on the *repository resource*.

```
resource-link-registry: - resource: repository link-relation: 'http://identifiers.emc.com/linkrel/alias' uri-template: X_ALIAS_SE
```

There are other settings within the YAML file for REST extensibility. Please refer to its comment documentation or Documentum REST development guide for further learning.

We are done here for the sample project exploring. In next post, we will add an additional operation to create alias sets in the sample project.



Documentum REST Extensibility Tutorial (3): Explore The Sample Project

**Next:** [Documentum REST Extensibility Tutorial \(4\): Create Object](#)

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