

Advanced Extensibility in Documentum REST 7.3

In Documentum REST Services 7.2, we added [Documentum REST Extensibility](#), by which users can create custom REST resources based on the Core REST offerings. We also provided a YAML configuration to users to add new root services under Home Document or add link relations onto existing Core resources.

Now in Documentum REST Services 7.3, the extensibility has been enhanced in a lot of areas. In this post, we will highlight significant new features and changes. The topic covers these items:

- Customizing YAML location
- Disabling Core resources
- Disabling XML or JSON media type
- Substituting Core resource representations
- Building link relations
- Marshalling custom resource models
- Upgrading to Spring Framework 4
- Transiting to Spring Java first configuration
- More SDK samples

Customizing YAML location

A majority of Documentum REST extensibility configurations are available via the YAML configuration. The default YAML file is located at `dctm-rest.war\WEB-INF\classes\com\lemc\documentum\rest\script\rest-api-custom-resource-registry.yaml`. With the YAML configuration, users can customize the REST services in these areas:

- Add new URI templates
- Register root services
- Add link relations onto Core resources
- Disable resources (7.3 new)
- Disable media types (7.3 new)
- Substitute resource views (representations, 7.3 new)

We will cover 7.3 new configurations later. But here we introduce a new runtime property configuration to relocate the YAML file. In a custom REST project, you may want to define the YAML file in your own path and rename it with your favorite. It's possible now in REST 7.3. In *dctm-rest.war\WEB-INF\classes\rest-api-runtime.properties*, you can specify a different location for the YAML file.

```
# the sample specifies the YAML file location at \WEB-INF\classes\com\acme\rest\config rest.ext.yaml.package = com.acme.rest.co
```

REST server runtime looks for the file with ***.yaml** name pattern. Please make sure your YAML file is end with **.yaml**.

In case you have multiple YAML files in this location, or you have a different file extension for the YAML file. You can set another property in *rest-api-runtime.properties* file, as below:

```
# the sample specifies the YAML file name my_custom_rest.yml under the YAML location rest.ext.yaml.name = my_custom_rest.y
```

With this configuration, you can create multiple YAML file templates during the development phase, and switch the YAML file from development stage to production stage by environment variables. No doubt that you can set REST runtime properties in environment variables of the server startup command.

Disabling Core resources

Documentum REST Services in release 7.3 provides 70+ resource APIs. They are useful by most content management applications. However, for the particular provisioned REST service, users may need to disable some of them for business requirement, for instance, disabling API access for type(s) resource, and relation(s) resource. In 7.3, we make this customization available in the YAML configuration.

```
# disables types, type, relations, and relation resources disabling-resource-registry: - resources: [types,type,relations,relation]
```

When the specified resources are disabled, getting these resources will result in status code **404 Not Found**. Please note disabling resources has the side effect on other resources. One example is, before the type resource disabling, a document resource has a link to the *dm_document* type resource.

```
{ "name": "document", "type": "dm_esign_template", "definition": "http://localhost:8080/dctm-rest/repositories/REPO/types/dm_esign_template"
```

Now after the type resource is disabled, the type definition link is auto removed from the representation by server.

```
{ "name": "document", "type": "dm_esign_template", "properties": { "object_name": "Default Signature Page Template", ... }
```

Documentum REST Services Development Guide has the detail explanation of the impact on disabling resources.

Disabling XML or JSON media type

Documentum REST Services supports both XML and JSON formats by default. In a custom REST service project, users may want to support only one kind of media type for its client applications. For instance, most JavaScript clients have better support for JSON message processing. To reduce the custom REST development maintenance effort, you can disable one media type in the YAML configuration.

```
# defaults to support both XML and JSON media-type-registry: - media-type: default
```

supports XML only media-type-registry: - media-type: xml

supports JSON only media-type-registry: - media-type: json

When **json** is set, getting a resource for XML media type will get status code **406 Not Acceptable**, and putting or posting a resource for XML media type will get status code **415 Unsupported Media Type** errors. For instance, the below sample shows the error for getting repositories by XML.

```
GET http://localhost:8080/dctm-rest/repositories Accept: */*+xml ----- HTTP/1.1 406 Not Acceptable
```

Substituting Core resource representations

Documentum REST Extensibility introduced [resource view](#) to compose the resource representations. In a custom REST project, users may want to customize Core resource representations for reasons like hiding some link relations, injecting some additional attributes and so on. The extensibility provides users a way to customize Core resource representations by substituting the resource view implementations with the YAML configuration.

For example, if you want to hide all internal attributes (starting with *i_*) in the cabinet resource, you can make customizations as below.

Creating custom cabinet view

```
public class CabinetViewBeta extends CabinetView {    public CabinetViewBeta2(...) {        super(...);    }    @Override    public void ...
```

Configuring YAML entry

```
resource-view-registry: - resource: cabinet    view:    [com.acme.view.impl.CabinetViewBeta]
```

By these two steps, the Core cabinet resource won't return any internal attributes in the XML and JSON representations any longer. Please check the *Java doc* for what APIs users can override and *Documentum REST SDK* for the runnable project sample.

Building link relations

Before release 7.3, users build a link relation for a custom resource by two steps:

1. Define a URI template in YAML file
2. Call `com.emc.documentum.rest.http.UriFactory` to build the link by URI template name

Since from release 7.3, the link build in REST code has been greatly improved, by the new class `com.emc.documentum.rest.context.ResourceUriBuilder`. Here are a few samples showing how easy it is to create links in Java code.

```
// build user resource URI: <base uri context>/repositories/<repoName>/users/dmadmin String href = ResourceUriBuilder.onResourceUriBuilder().buildUri("users/dmadmin")
```

```
// build query resource URI as hreftemplate: <base uri context>/repositories/<repoName>{?q} String href = ResourceUriBuilder.onResourceUriBuilder().buildUri("repositories/{repoName}/{?q}")
```

```
// build custom alias-set resource URI by resource controller: <base uri context>/repositories/<repoName>/alias-set/1234?view=:all String href = ResourceUriBuilder.onResourceUriBuilder().buildUri("repositories/{repoName}/alias-set/1234?view=:all")
```

```
// build custom alias-set resource URI by custom URI template: <base uri context>/repositories/<repoName>/alias-set/1234?view=:all String href = ResourceUriBuilder.onResourceUriBuilder().buildUri("repositories/{repoName}/alias-set/1234?view=:all")
```

Please checkout the Java doc of this class for its usage.

Marshalling custom resource models

Documentum REST Services provides its own marshalling framework to serialize and deserialize Java resource models to/from XML and JSON messages, with a small set of Java annotations (e.g. `@SerializableType`, `@SerializableField`). Developers only need to focus on resource model design at the development phase. In release 7.3, the marshalling framework is more open to developers. A lot of improvements are made in these areas:

- Support Java Map data type
- Support generic, wild, abstract and interface types
- Support resource arbitrary data type field customization for serialization and deserialization, respectively

Here is a simple example for Java Map support.

Creating a resource model

```
@SerializableType("bom") public class BusinessObjectMap { Map<String, String> codes; ... } BusinessObjectMap bom = new Bus
```

Marshalling to JSON

```
{ "codes":{ "usa":"001", "china":"086" } }
```

Marshalling to XML

```
<bom> <codes> <usa>001</usa> <china>086</china> </codes> </bom>
```

You can also marshal the resource model in different ways by deeply configuring the annotations.

Documentum REST Services Development Guide and *SDK samples* provides the detail information about the marshalling framework features.

Upgrading to Spring Framework 4

As many of you know, Documentum REST Services is building upon Spring MVC, so we are heavily using Spring Framework technologies. Documentum REST release 7.2 uses Spring Framework 3.2.x. Per Spring team's [announcement](#), Spring 3.2.x goes to end of life on Dec 31, 2016. And Spring 4.3.x is the [recommendation](#) before Spring 5 GA. For that reason, we decided to upgrade to Spring 4.3.x line in Documentum REST Services 7.3. We also upgraded Spring Security 4.1.x line for the authentication development. Spring Security 4.x works better with Spring Framework 4.x, and it brings a lot of security enhancements that are used in Documentum REST 7.3 new feature development.

The impact to custom REST projects are, when upgrading Documentum REST artifacts to 7.3, you will need to upgrade to Spring Framework 4 and Spring Security 4 as well. Please refer to [What's New in Spring Framework 4](#) and [What's New in Spring Security 4](#) for details.

Transiting to Spring Java first configuration

By Documentum REST Services 7.2, we use Spring XML configuration to configure beans and security for the REST services and authentications. As we expose deeper customizations to REST users, XML configuration turns out to be limited in a lot of areas. As Spring 4 made a lot of improvements on Java configuration, we finally decided to completely move to Java configuration for all Spring configurations.

Here is an example to compare XML configuration and Java configuration in Documentum REST services.

XML configuration in Documentum REST 7.2

```
<beans xmlns="http://www.springframework.org/schema/beans"      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

Java configuration in Documentum REST 7.3

```
@Configuration @ComponentScan(basePackages = "com.acme.sample",      excludeFilters = { @ComponentScan.Filter(type = F
```

Moreover, we expose authentication extensibility in Documentum REST Services 7.3, in which all security configurations are Java based. *Documentum REST Services Development Guide* has more details on the Java configuration. All SDK samples and archetype project are updated with Java configuration, too.

For the backward compatibility consideration, custom REST projects can still use XML configuration in the new services, but it is ***strongly recommended*** to transit to Java configuration soon.

More SDK samples

In Documentum REST Services 7.3 SDK, we add more samples on creating new resources, customizing Core resources and adding new authentication schemes. Please check out the SDK zip for your references. The REST team will add more samples based on your feedback.

