

Documentum TBO, SBO, Aspect and REST Extensibility, Part II

In this series, we will walk through Documentum full-stack extensibility from Business Object Framework (BOF) to REST API development. The tutorials are divided into 3 parts:

- [Part I](#): Design user story and install BOF
- [Part II](#): Consume Type-based Business Object (TBO) / Aspect by default REST API
- [Part III](#): Design fine-grained REST API for TBO / Service-based Business Object (SBO) / Aspect

This article is the 2nd part of the series, talking about how to consume custom type and TBO in the out-of-the-box REST API.

Preliminary

Please make sure you have installed all the artifacts following [Part I: Design user story and install BOF](#).

Tools we'll use in this part:

- Documentum Content Server 7.1+ with installed BOF artifacts
- Documentum REST Server 7.3+
- Postman

[Postman](#) is a nice & free REST client tool to debug the REST API. There are some other alternatives but the scripts we make in this tutorial will be in Postman format.

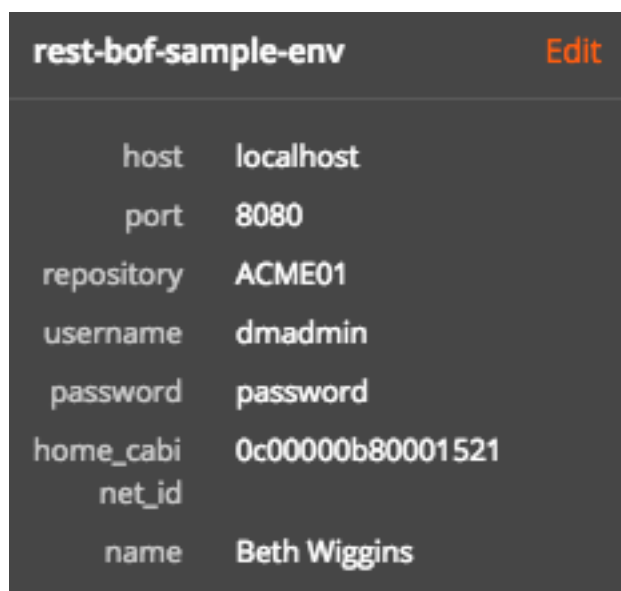
The Postman script files can be downloaded from GitHub: [documentum-rest-extensibility-samples/rest-bof-sample/postman-test at master · Enterprise-Content-Management/documentum-...](#)

Deploy Documentum REST Services

You can follow **Documentum REST Install Guide** to deploy the REST service. If this is your first time to install Documentum REST, there are two articles that you can refer to: [How to install Documentum Core REST](#), [Documentum REST Tutorial](#).

Prepare Postman scripts

Please get the Postman script files from GitHub, then open Postman, import the script file **REST-BOF-Tutorial.collection.json** into Postman **Collections**, and import the script file **REST-BOF-Tutorial.env.json** into Postman **Environments**. Before you invoke the requests, you need to update the REST host and repository environment variables in Postman Environment. Here are the default ones.



rest-bof-sample-env		Edit
host	localhost	
port	8080	
repository	ACME01	
username	dmadmin	
password	password	
home_cabi	0c00000b80001521	
net_id		
name	Beth Wiggins	

Explore custom types with REST API

With the environment prepared, now we can test the custom types we have just installed by the out-of-the-box REST API.

a1. Get type name_card

If we send request for **a1**, it will return us the **name_card** type definition. Here is the REST message dump.

Request:

```
GET /dctm-rest/repositories/ACME01/types/name_card?inherited=false HTTP/1.1 Host: localhost:8080 Authorization: Basic ZG1hZG1p
```

Response:

```
{  "name": "name_card",  "label": "Name Card",  "category": "standard",  "parent": "http://localhost:8080/dctm-rest/repositories/ACME01/types/name_card"
```

a2. Get aspect type contact_aspect

If we send request for **a2**, it will return us the **contact_aspect** type definition. Here is the REST message dump.

Request:

```
GET /dctm-rest/repositories/ACME01/aspect-types/contact_aspect HTTP/1.1 Host: localhost:8080 Authorization: Basic ZG1hZG1p
```

Response:

```
{  "name": "aspect-type",  "type": "dmc_aspect_type",  "definition": "http://localhost:8080/dctm-rest/repositories/ACME01/types/aspect-type"
```

a3. Get aspect type organization_aspect

It's very similar to a2.

Create custom type instances with REST API

One big question is, can we create custom type objects and trigger the TBO automatically by the out-of-the-box REST API? The answer is YES. But there are some restrictions.

- Custom object type instances can be created by the default REST API only when the custom type is a sub type supported by the REST API, i.e. sub types of **dm_document** instances can be created under **Folder Child Documents Resource**, and sub types of **dm_user** instances can be created under **Users Resource**.
- Custom object type's TBO can be triggered if the REST API implementation has called corresponding overridden DFC methods. For instance, TBO will be triggered on object **creation** and **update** if the method **doSave()** is overridden in TBO class. But TBO won't be triggered on content download.

In this tutorial, **NameCard** class in the TBO has overridden **doSave()** method. Thus, on the **name_card** object creation, the TBO will be triggered automatically. As we know, **name_card** type is a sub type of **dm_document**, so we should be able to create **name_card** objects as regular **dm_document** objects.

In Documentum REST Services, document creation is usually performed under **Folder Child Objects Resource**, or **Folder Child Documents Resource**, seeing [Tutorial: Manage Document Contents in REST Services](#). In the Postman scripts, steps **a4. Get home cabinet** can help us find the link relation for document creation. Here we jump to the document creation request directly.

b1. Create contentless name card

On a regular folder, we can create **name_card** objects under it. The sample **b1** creates a contentless name card. The

Request:

POST /dctm-rest/repositories/ACME01/folders/0c00000b80001521/documents HTTP/1.1 Host: localhost:8080 Content-Type: application/json

Response:

```
{  "name": "document",  "type": "name_card",  "definition": "http://localhost:8080/dctm-rest/repositories/ACME01/types/name_card"
```

The response is a regular document object representation. As you observed, the response indicates that **has_contact=true** and **has_organization=true**. This means TBO is triggered because they are set to true only by the TBO class.

b2. Create contentful name card

In the last sample, the newly created name card does not have a photo or a biography for the user. This sample illustrates how to create a contentful name card object under the same parent resource. The request is a HTTP multipart request:

- Part 1: JSON object properties
- Part 2: text content for biography
- Part 3: photo in PNG format

Request:

POST /dctm-rest/repositories/ACME01/folders/0c00000b80001521/documents?content-count=2&all-primary=false&forma

Response:

```
{  "name": "document",  "type": "name_card",  "definition": "http://localhost:8080/dctm-rest/repositories/ACME01/types/name_c
```

Please note the property **r_content_size=49**, which indicates this instance has contents. We can follow the link relation **contents** to get its content metadata collection and download the content binary.

By now, we have gotten to know how to create a custom type object with the out-of-the-box REST API and in which conditions TBOs will be triggered.

Get custom type instances with REST API

Since this is the out-of-the-box REST API, there is not a dedicated collection to return all name card instances. But with the REST API, we can still get them by filtering the result.

Request:

GET /dctm-rest/repositories/ACME01/folders/0c00000b80001521/documents?object-type=name_card HTTP/1.1 Host: localhost:8080

Response:

```
{  "id": "http://localhost:8080/dctm-rest/repositories/ACME01/folders/0c00000b80001521/documents",  "title": "Documents under"
```

The result shows that there are 2 name card instances.

Summary

In this tutorial, we learnt how to get custom types and CRUD custom type objects by the out-of-the-box REST API. It works well but it's not perfect. For instance,

- The newly created name card response did not return the link relations to biography and photo directly
- The filtered documents collection returns some entry metadata that we don't want to see, i.e. "summary": "name_card 0900000b80037e7c".
- The object creation does not prevent creating name card instances with the same user name.
- The object creation does not allow setting aspect properties.

To address all these concerns, we need to create custom REST services for the dedicated object type. Please read further on part III.

Part III: Design fine-grained REST API for TBO / Service-based Business Object (SBO) / Aspect

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