Restful Objects Viewer

Table of Contents

Chapter 1. Restful Objects Viewer

Apache Isis' Restful Objects viewer is an implementation of the Restful Objects spec, which defines a generic way to expose a domain model through a REST (or more precisely, hypermedia) API. Having a REST API opens up an Apache Isis domain model to a huge variety of applications, from bespoke single-page apps, through integration scenarious, through providing an API for bulk-upload/migration from an existing system.

The Restful Objects viewer also provides a number of extensions specific to Apache Isis. Most significant of these is enhanced content negotiation support, making it easier to use the returned representations within bespoke clients using standard third-party configurations.

This user guide discuss features, configuration and also how to extend the Restful Objects viewer.

1.1. Other Guides

Apache Isis documentation is broken out into a number of user, reference and "supporting procedures" guides.

The user guides available are:

- Fundamentals
- Wicket viewer
- Restful Objects viewer (this guide)
- DataNucleus object store
- Security
- Testing
- Beyond the Basics

The reference guides are:

- Annotations
- Domain Services
- Configuration Properties
- Classes, Methods and Schema
- Apache Isis Maven plugin
- Framework Internal Services

The remaining guides are:

- Developers' Guide (how to set up a development environment for Apache Isis and contribute back to the project)
- Committers' Guide (release procedures and related practices)

Chapter 2. RO Specification

The Restful Objects v1.0 specification defines a comprehensive hypermedia API, consisting of HTTP resources and corresponding JSON representations, for accessing and manipulating a domain object model.

The Restful Objects spec can be downloaded from here as either a PDF or Word doc.

2.1. Goals of the Spec

The goal of Restful Objects is to allow domain models to be accessed through HTTP resources, returning a set of JSON representations. These representations can then be consumed by any client (e.g. Javascript, Java, .NET, Ruby, Python).

Both the resources and representations are generalized so that they can be applied to any domain model, and by default all representations have media types designed to allow a completely generic client to be written, capable of working, unmodified, with any domain model that has a Restful Objects interface.

Alternatively, the developer may write a custom client that has some shared knowledge of the domain being exposed, and render the information in a more specific fashion.

Restful Objects also defines that representations are served up with parameterized media types. This allows clients to use content negotiation to ensure that representations do not change in a breaking fashion, enabling server and client to evolve independently.

The Restful Objects specification is at a higher-level of abstraction than, say, the JAX-RS specifications for Java platform, or the WCF specifications on .NET. Specifically, the domain classes that it exposes are represented in a very general form. They consist of:

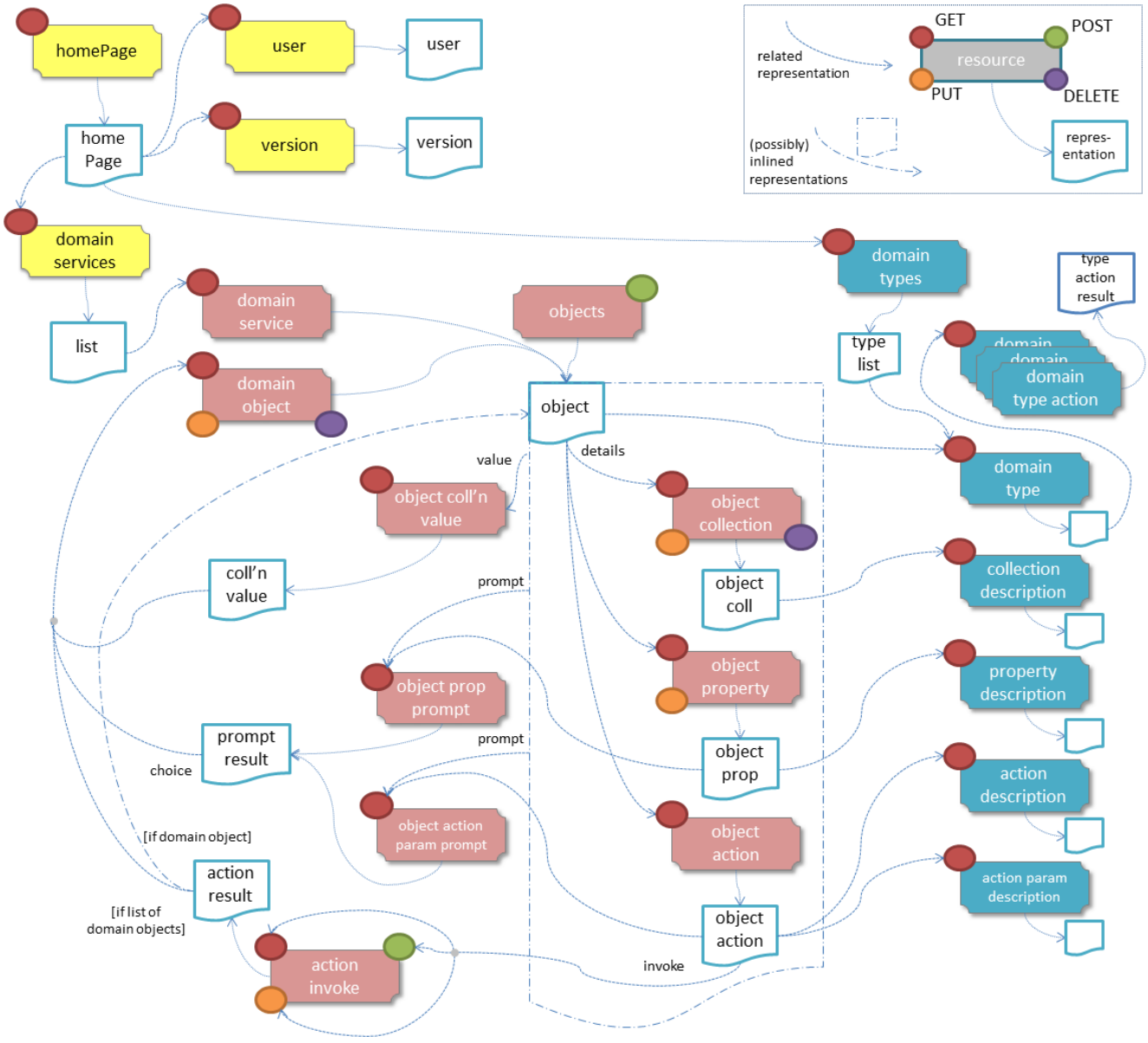
- properties (fields), each holding either a scalar value or reference to another object;
- collections, each holding a vector reference to other entities;
- actions (operations/methods), whereby the object can execute business logic.

Beyond this, though, Restful Objects makes very few assumptions. In particular, Restful Objects does not prescribe the nature of the domain model.

The Restful Objects spec may be downloaded directly from github as either a PDF or as a Word doc.

2.2. Resources and Representations

The diagram below - taken from the Restful Objects spec - shows the various resources (URLs) and representations (JSON) that are defined:



The resource URLs are always well-defined, but Apache Isis' implementation allows for different representations to be returned, using content negotiation. This is discussed further below.

In fact, there's nothing in Apache Isis to prevent you from defining your own REST controllers to provide custom resource URLs. One use case might be to support user registration/authentication, a topic out-of-scope of the RO spec itself.

2.3. Apache Isis' implementation

The Restful Objects viewer is Apache Isis' implementation of the Restful Objects spec. It implements all the mandatory features of the specification. It also implements some of the optional capabilities (as defined in section 3 of the RO spec, and as represented in the version resource, section 8.1):

Capability	Support	Notes
blobsClobs	yes	
deleteObjects	yes	

Capability	Support	Notes
domainModel	formal	The 'simple' scheme is *not* supported
validateOnly	yes	
protoPersistentObjects	yes	

2.4. Extensions

The Restful Objects viewer also implements some of the "future ideas" that are out of scope for the RO spec v1.0, but described in section 34, "ideas for future extensions".

2.4.1. Content Negotiation (34.1)

Apache Isis provides two levels of support for content negotiation.

`x-ro-domain-type`

The first level is very similar to the "Domain Model Agnostic" approach sketched out in the RO spec. The client can send an `x-ro-domain-type` parameter for either domain object representations (section 14 of the spec) or action invocation results (section 19 of the spec). This can be combined with either `application/json` or `application/xml`.

For example, the client could use an `Accept` header such as:

```
Accept: application/xml;x-ro-domain-
type="com.mycompany.viewmodels.v2.CustomerViewModel"
```

The server will use the `ContentMappingService` to attempt to transform the domain object into the requested `x-ro-domain-type`. The whole process is discussed in more detail in the architecture chapter.

Apache Isis profile

The representations defined by the RO spec are very rich and enable complex client-side applications to be built. However, their sophistication can be an impediment to their use if one wishes to write a simple app using third-party components that expect to consume much simpler representations. Examples of such tools are Angular Bootstrap, Angular XEditable, Angular Strap.

This support is discussed further in the simplified representations chapter.

2.4.2. Minimizing Round-trips (34.4)

The Restful Objects viewer supports the `x-ro-follow-links` query parameter in a way very similar to that suggested in the RO spec, the main point being to avoid the "N+1" problem of too many (slow) network calls. For example, using this feature one can load a grid of data in a single call. (That said, the simplified representations supported by Restful Objects viewer also support this use case,

albeit in way that deviates from the RO spec).

This screencast demonstrates the Restful Object viewer's support for `x-ro-follow-links` parameter, using the (non-ASF) Isis addons' kitchensink app as the example, This app contains three entities, `Grandparent`, `Parent` and `Child` that define a hierarchy of 1:m relationships.

The queries that are shown in the screencast include:

- show parent and its children (titles)

  <pre>http://localhost:8080/restful/objects/PARENT/0?x-ro-follow-links=members[children].value

- show parent and its children (full details)

  <pre>http://localhost:8080/restful/objects/PARENT/0?x-ro-follow-links=members[children].value.href

- child's parent (title)

  <pre>http://localhost:8080/restful/objects/CHILD/0?x-ro-follow-links=members[parent].value

- child's siblings (up to its parent, down to children)

  <pre>http://localhost:8080/restful/objects/CHILD/0?x-ro-follow-links=members[parent].value.members[children].value

Honor UI hints

By default the representations generated by Restful Objects ignore any Apache Isis metamodel hints referring to the UI. In particular, if a collection is annotated then `Render(EAGERLY)` then the contents of the collection are *not* eagerly embedded in the object representation.

However, this behaviour can be overridden globally using following property (typically added to `WEB-INF/viewer_restfulobjects.properties`):

```
isis.viewer.restfulobjects.honorUiHints=true
```

This means that standard Apache Isis annotations can be used as a simple way to obtain follow-links (driven from the server model, though, rather than the requesting client).

Chapter 3. Architecture

The RestfulObjects viewer implements the Restful Object spec, meaning that it defines a well-defined set of endpoint URLs as resources, and generates a well-defined set of (JSON) representations when these resources are accessed.

By default, the Restful Objects viewer will automatically handle requests and return representations according to the RO spec. However, its internal architecture provides several hooks for content negotiation, thereby allowing the generated representation to be influenced using the standard HTTP `Accept` header. In response, the server uses the `Content-Type` header which the client can use to know how to process the returned representation.

- `RepresentationService`

  The `RepresentationService` is an SPI domain service (plugin-point) that allows an arbitrary representation to be generated for any of the resources defined in the RO spec.

  Normally this SPI service need not be replaced, because the default implementation (`RepresentationServiceContentNegotiator`) simply uses the HTTP `Accept` header and delegates onto another service, the (slightly misnamed) `ContentNegotiationService`, to actually generate the representation. There can be multiple implementations of the `ContentNegotiationService` and the content negotiator will delegate to each in turn until one is able to handle the request (per the chain of responsibliity pattern).

- `ContentNegotiationService`

  As noted above, there can be multiple implementations of the `ContentNegotiationService`, each one handling a particular HTTP `Accept` header. If the implementation does not recognize the value of the header, if can simply return `null`.

  The framework provides a number of implementations; an implementation that handles the simplified representation of the Apache Isis profile; an implementation that provides support for the `x-ro-domain-type` parameter, and a default/fallback implementation that returns the representations defined by the RO spec.

- `ContentMappingService`

  The `ContentMappingService` is used by the implementation of `ContentNegotationService` that recognizes the `x-ro-domain-type`, its role being to transform a domain object (usually an entity) into some other form (usually a DTO), as specified by the `x-ro-domain-type` parameter. There can be many such implementations, each handling a different target domain type.

This diagram shows how these services collaborate:

Taken together these domain services offer a lot of flexibility in terms of the representations that can be generated from the RestfulObjects viewer.

Chapter 4. Layout Resources (1.16.0-SNAPSHOT)

(As of 1.16.0-SNAPSHOT) Apache Isis' Restful Objects viewer provides a number of additional resource endpoints that provide representations of the object layout (as per GridService) and of the menu layout (as per MenuBarsService).

This chapter provides details of these resources, the link Rels to access them, and the resultant representations.

4.1. MenuBars

The MenuBarsService provides the menu.layout.xml XML document which defines how to group the various domain service actions into menubars, menus and menu sections.

For example, the Hello World archetype has the following layout:

```
<mb3:menuBars
    xsi:schemaLocation="..."
    xmlns:cpt="http://isis.apache.org/applib/layout/component"
    xmlns:lnk="http://isis.apache.org/applib/layout/links"
    xmlns:mb3="http://isis.apache.org/applib/layout/menubars/bootstrap3"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <mb3:primary>
        <mb3:menu>
            <mb3:named>Hello World Objects</mb3:named>
            <mb3:section>
                <mb3:serviceAction objectType="helloworld.HelloWorldObjects" id=
"create">
                    <cpt:named>Create</cpt:named>
                </mb3:serviceAction>
                ...
            </mb3:section>
        </mb3:menu>
        <mb3:menu unreferencedActions="true">
            <mb3:named>Other</mb3:named>
        </mb3:menu>
        ...
    </mb3:primary>
    <mb3:secondary>
        ...
    </mb3:secondary>
    <mb3:tertiary>
        ...
    </mb3:tertiary>
</mb3:menuBars>
```

Note that exactly one `<mb3:menu>` must have the `unreferencedActions` flag set. Any service actions that are not explicitly listed will be added to this menu.

The representation returned by home page resource (section 5.2 of the RO spec v1.0) has been extended to provide a link to this resource:

```
{
  "links": [
    ...
    {
      "rel": "urn:org.apache.isis.restfulobjects:rels/menuBars",
      "href": "http://localhost:8080/restful/menuBars",
      "method": "GET",
      "type": "application/json;profile='urn:org.restfulobjects:repr-types/layout-
menubars'"
    },
  ],
  ...
}
```

The representation returned by the `/menuBars` resource (assuming an HTTP header of `Accept: application/xml`) is a superset of the `menu.layout.xml`; each action also includes a `link` to the corresponding Restful Objects resource:

```
<mb3:serviceAction objectType="helloworld.HelloWorldObjects" id="create">
  <cpt:named>Create</cpt:named>
    <cpt:link>
      <lnk:rel>urn:org.restfulobjects:rels/action</lnk:rel>
        <lnk:method>GET</lnk:method>
        <lnk:href>

http://localhost:8080/restful/objects/helloworld.HelloWorldObjects/1/actions/create
        </lnk:href>
        <lnk:type>
          application/json;profile="urn:org.restfulobjects:repr-types/object-action"
        </lnk:type>
  </cpt:link>
</mb3:serviceAction>
```

This can also be obtained in JSON format in the usual way (by specifying an HTTP header of `Accept: application/json`):

```
"serviceAction": [
{
  "objectType": "helloworld.HelloWorldObjects",
  "id": "create",
  "named": "Create",
  "link": {
    "rel": "urn:org.restfulobjects:rels/action",
    "method": "GET",
    "href":
"http://localhost:8080/restful/objects/helloworld.HelloWorldObjects/1/actions/create",
    "type": "application/json;profile=\"urn:org.restfulobjects:repr-types/object-
action\""
  }
}
```

4.2. Domain Object Layout

The `GridService` provides an XML document which defines the layout of any of domain object. Typically this is the contents of the `Xxx.layout.xml` file (where `Xxx` is the domain type).

For example, in the Hello World archetype the `HelloWorld` domain object has a layout defined by `HelloWorld.layout.xml`.

The representation returned by the domain object resource (section 14.4 of the RO spec v1.0) has been extended to provide a link to this resource:

```
{
  "links": [
    ...
    {
      "rel": "urn:org.apache.isis.restfulobjects:rels/object-layout",
      "href":
"http://localhost:8080/restful/objects/helloworld.HelloWorldObject/0/object-layout",
      "method": "GET",
      "type": "application/json;profile='urn:org.restfulobjects:repr-types/object'",
      "title": "Object: a"
    },
  ],
  ...
}
```

In a similar way to the `menu.layout.xml`, the representations is supplemented with `link`s nodes that link back to the standard Restful Objects resources:

- `domainObject`
- `property`
- `collection`

- `action`

For example, the layout for a "HelloWorldObject" instance in the hello world archetype (with `Accept: appication/xml` HTTP header) is:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bs3:grid xmlns:cpt="http://isis.apache.org/applib/layout/component"
          xmlns:lnk="http://isis.apache.org/applib/layout/links"
          xmlns:bs3="http://isis.apache.org/applib/layout/grid/bootstrap3">
  <bs3:row>
    <bs3:col span="12" unreferencedActions="true">
      <cpt:domainObject bookmarking="AS_ROOT">
          <cpt:link>
          <lnk:rel>urn:org.restfulobjects:rels/element</lnk:rel>
          <lnk:method>GET</lnk:method>

<lnk:href>http://localhost:8080/restful/objects/helloworld.HelloWorldObject/0</lnk:hre
f>
          <lnk:type>application/json;profile="urn:org.restfulobjects:repr-
types/object"</lnk:type>
        </cpt:link>
      </cpt:domainObject>
    </bs3:col>
  </bs3:row>
  ...
</bs3:grid>
```

This can also be obtained as JSON (using `Accept: application/json` HTTP header):

```
{
  "row": [
    {
      "cols": [
        {
          "col": {
            "domainObject": {
              "link": {
                "rel": "urn:org.restfulobjects:rels/element",
                "method": "GET",
                "href":
"http://localhost:8080/restful/objects/helloworld.HelloWorldObject/0",
                "type": "application/json;profile='urn:org.restfulobjects:repr-
types/object'"
              },
              "bookmarking": "AS_ROOT",
            },
            "span": 12,
            "unreferencedActions": true
          }
        }
      ]
    }
    ...
  ]
}
```

4.3. Domain Type Layout

The representation of the domain types resource (section 22.2 of RO spec v1.0) has also been extended to return the (type) layout:

```
{
  "links": [
    ...
    {
      "rel": "urn:org.apache.isis.restfulobjects:rels/layout",
      "href": "http://localhost:8080/restful/domain-
types/helloworld.HelloWorldObject/layout",
      "method": "GET",
      "type": "application/json;profile='urn:org.restfulobjects:repr-types/layout-
bs3'"
    }
  ],
  ...
}
```

The representation returned by this resource is essentially exactly the same as the layout returned

by `GridService` (it is not dynamically extended with links).

4.4. Static vs Dynamic Resources

The menu layout representation includes all possible domain services; it does not follow that the current user has access to all of these actions (some may be hidden or disabled).

Similarly, the domain object layout representation include all _possible properties, collections and actions of the domain object; again, the current user may not have access to all of these members. It is also often the case that the domain object’s internal state will determine which members to make available (eg, show only one of "lock" and "unlock" actions at any given time).

To determine what should actually be rendered, the REST client should follow the links to the standard Restful Objects resources.

Chapter 5. Simplified Representations

The representations defined by the RO spec are very rich and enable complex client-side applications to be built. However, their sophistication can be an impediment to their use if one wishes to write a simple app using third-party components that expect to consume much simpler representations. Examples of such tools are Angular Bootstrap, Angular XEditable, Angular Strap.

As of `1.11.0`, Apache Isis provides support for its own simplified representation for the most commonly-used representations. This is implemented using the `ContentNegotiationService` described in the architecture chapter.

5.1. The Apache Isis "Profile"

The RO spec uses the standard `Accept` header for content negotiation, and defines its own "profile" for the standard representations; these take the form:

```
Accept: application/json;profile="urn:org.restfulobjects:repr-types/xxx"
```

where "xxx" varies by resource. The detail can be found in section 2.4.1 of the RO spec.

The Apache Isis viewer also defines its own "Isis" profile which enables the client to request simplified representations for the most frequently accessed resources. This is done by specifying an `Accept` header of:

```
Accept: application/json;profile="urn:org.apache.isis/v1"
```

Not every resource supports this header, but the most commonly accessed ones do. In each case the server will set the `Content-Type` header so that the client knows how to process the representation.

The screencast demonstrates the feature.

The sections below explain in a little more detail what is returned when this profile is activated.

5.2. Domain Object

If a domain object resource (section 14) is accessed with the Apache Isis profile, the resultant representation is a JSON object with simple key/value pairs for each property.

The contents of any collections are also eagerly returned, consisting of an array of elements of each referenced object. Each such element contains key/value pairs of each property (in other words, a grid of data is returned). Each element also has a special `href` property (so that the client can easily navigate to a resource for that object) and a `title` property (to use as a label, eg the hyperlink text).

In addition, the representation defined by the RO spec is also included, under a special `$$ro`

property.

For example, using the (non-ASF) Isis addons' todoapp, accessing this resource:

```
http://localhost:8080/restful/objects/TODO/45
```

with an `Accept` request header of:

```
Accept: application/json;profile="urn:org.apache.isis/v1"
```

returns the following representation:

```
{
  "$$href" : "http://localhost:8080/restful/objects/TODO/45",       ①
  "$$instanceId" : "45",                                            ②
  "$$title" : "Buy bread due by 2015-12-04",                        ③
  "description" : "Buy bread",                                      ④
  "category" : "Domestic",
  "subcategory" : "Shopping",
  "complete" : false,
  "atPath" : "/users/sven",
  ...
  "similarTo" : [ {                                                 ⑤
    "$$href" : "http://localhost:8080/restful/objects/TODO/46",
    "$$instanceId" : "46",
    "$$title" : "Buy milk due by 2015-12-04",
    "description" : "Buy milk",
    "category" : "Domestic",
    ...
  }, {
    "$$href" : "http://localhost:8080/restful/objects/TODO/47",
    "$$instanceId" : "47",
    "$$title" : "Buy stamps due by 2015-12-04",
    "description" : "Buy stamps",
    "category" : "Domestic",
    ...
  },
     ...
  } ],
  "dependencies" : [ ],
  "$$ro" : {                                                        ⑥
    "links" : [ ... ],
    "extensions" : { ... },
    "title" : "Buy bread due by 2015-12-04",
    "domainType" : "TODO",
    "instanceId" : "45",
    "members" : { ... }
  }
}
```

① hyperlink to the representation

② instance id of the domain object (unique within its type)

③ title of the domain object

④ all the properties of the domain object (to which the caller has access), as key/value pairs

⑤ contents of each collection

⑥ special `$$ro` json-prop, being the normal RO Spec representation for this object

with a `Content-Type` header:

```
Content-Type: application/json;
              profile="urn:org.apache.isis/v1";repr-type="object"
```

5.3. Domain Object Collection

If a domain object collection (section 17) is accessed with this profile, then the resultant representation is as an array of elements of key/value for each referenced object, and again each element the containing the key/value pairs of the properties of that object (a grid, again).

In addition, the representation defined by the RO spec is also included, as a special object with a single `$$ro` property.

For example, using the (non-ASF) Isis addons' todoapp, accessing this resource:

```
http://localhost:8080/restful/objects/TODO/45/collections/similarTo
```

with an `Accept` request header of:

```
Accept: application/json;profile="urn:org.apache.isis/v1"
```

returns the following representation:

```
[                                                                    ①
{
  "$$href" : "http://localhost:8080/restful/objects/TODO/46",        ②
  "$$instanceId" : "46",                                             ③
  "$$title" : "Buy milk due by 2015-12-04",                          ④
  "description" : "Buy milk",                                        ⑤
  "category" : "Domestic",
  ...
}, {
  "$$href" : "http://localhost:8080/restful/objects/TODO/47",
  "$$title" : "Buy stamps due by 2015-12-04",
  "description" : "Buy stamps",
  "category" : "Domestic",
  ...
}, {
  "$$href" : "http://localhost:8080/restful/objects/TODO/48",
  "$$title" : "Mow lawn due by 2015-12-10",
  "description" : "Mow lawn",
  "category" : "Domestic",
  ...
},
...
, {
  "$$ro" : {                                                         ⑥
    "id" : "similarTo",
    "memberType" : "collection",
    "links" : [ ... ],
    "extensions" : { ... },
    "value" : [ ... ]
  }
}
]
```

① returns a JSON array, not a JSON object

② hyperlink to the representation

③ instance id of the domain object (unique within its type)

④ title of the domain object

⑤ all the properties of the domain object (to which the caller has access), as key/value pairs

⑥ last element is a special object with a single `$$ro` json-prop, being the normal RO Spec representation for this object

with a `Content-Type` header:

```
Content-Type: application/json;profile="urn:org.apache.isis/v1";repr-type="object-
collection"
```

5.4. Action Invocation

When an action is invoked, it can return a domain object, a list, a scalar, or return nothing.

5.4.1. Returning an Object

If the action returned an object, then the domain object representation described above is returned.

For example, using the (non-ASF) Isis addons' todoapp, accessing this resource:

```
http://localhost:8080/restful/objects/TODO/45/actions/updateCost/invoke
```

with an `Accept` request header of:

```
Accept: application/json;profile="urn:org.apache.isis/v1"
```

returns the following representation:

```
{
  "$$href" : "http://localhost:8080/restful/objects/TODO/45",
  "$$instanceId" : "45",
  "$$title" : "Buy bread due by 2015-12-04",
  "description" : "Buy bread",
  "category" : "Domestic",
  "subcategory" : "Shopping",
  "complete" : false,
  ...
  "similarTo" : [ ... ]
  ...
  "$$ro" : { ... }
}
```

with a `Content-Type` of:

```
Content-Type: application/json;profile="urn:org.apache.isis/v1";repr-type="object"
```

... in other words no different to a representation obtained of the returned domain object directly.

5.4.2. Returning a List

On the other hand if the action returned a list (a "standalone" collection, then an array representation is returned. This is very similar to that returned by a (parented) object collection, though with a slightly different `Content-Type` to distinguish.

For example, using the (non-ASF) Isis addons' todoapp, accessing this resource:

```
http://localhost:8080/restful/services/ToDoItems/actions/notYetComplete/invoke
```

with an `Accept` request header of:

```
Accept: application/json;profile="urn:org.apache.isis/v1"
```

returns the following representation:

```
[ {
  "$$href" : "http://localhost:8080/restful/objects/TODO/45",
  "$$instanceId" : "45",
  "$$title" : "Buy bread due by 2015-12-04",
  "description" : "Buy bread",
  "category" : "Domestic",
  ...
}, {
  "$$href" : "http://localhost:8080/restful/objects/TODO/46",
  "$$instanceId" : "46",
  "$$title" : "Buy milk due by 2015-12-04",
  "description" : "Buy milk",
  "category" : "Domestic",
  ...
},
...
, {
  "$$ro" : {
    "links" : [ ... ]
    "resulttype" : "list",
    "result" : { ... }
      "value" : [ ... ],
      "links" : [ ... ],
      "extensions" : { }
    }
  }
} ]
```

with a `Content-Type` header:

```
Content-Type: application/json;profile="urn:org.apache.isis/v1";repr-type="list"
```

5.4.3. Returning Scalar/Nothing

Note that actions returning scalar values or nothing (which includes `void` actions) are not supported; for these the regular RO spec representation will be returned.

5.5. Other Representations

Sometimes though you may want to extend or change the representations generated. This might be because you want to write a RESTful client that uses a particular library (say a Javascript library or web components) that can only handle representations in a certain form.

Or, you might want to have Apache Isis generate representations according to some other "standard", of which there are many:

- Mike Kelly's HAL specification
- Mike Amundsen's Collection+JSON specification
- Kevin Swiber's Siren specification
- Steve Klabnik's JSON API specification
- Gregg Cainus' Hyper+JSON specification
- the W3C's JSON-LD specification
- Markus Lanthaler's Hydra specification.

A good discussion about the relative merits of several of these different hypermedia formats can be found here.

Or, of course, you may have your own internal specification that you wish to use.

Supporting any of these alternative representations can be achieved by providing a suitable implementation of `ContentNegotiationService`. The existing implementations (eg `ContentNegotiationServiceSimplified`) can be used as a starting point.

These will, admittedly, need to access the internal APIs for the Apache Isis metamodel, and you should be aware that these are not formal API; they may change over time. That said, they are very stable and have not changed significantly over the last few years.

5.6. Global Config Props (Deprecated)

If all that is required is a very simple representations (of objects), you can configure the Restful Objects viewer to provide a simplified output, then this can be done with a number of (global) configuration properties.

These configuration properties pre-date the support, introduced in `1.11.0`, for the Apache Isis profile, and are limited by the fact that they are global configuration settings, so cannot be influenced on a request-by-request basis (as is the case with the `Accept` header used for the Apache Isis profile). They have therefore been deprecated, and may be removed in the future.

Details can be found in here.

Chapter 6. Configuration Properties

The Restful Objects viewer provides a couple of configuration option that extend/simplify/alter the representations generated from the Restful Objects specification.

These configuration properties are typically stored in `WEB-INF/viewer_restfulobjects.properties`. However, you can place all configuration properties into `WEB-INF/isis.properties` if you wish (the configuration properties from all config files are merged together).

6.1. Standard

The following configuration properties are supported:

Table 1. Restful Objects Viewer Configuration Properties

Property	**Value** (*default value*)	**Description**
`isis.viewer.restfulobjects.honorUiHints`	`true`,`false` (`false`)	A mechanism for reducing the number of round-trips by eagerly rendering collections; discussed here.
`isis.viewer.restfulobjects.strictAcceptChecking`	`true`,`false` (`false`)	Whether to strictly enforce the `Accept` header checking for the default RO-spec representations (by the `ContentNegotiationServiceForRestfulObjectsV1_0` service). Will otherwise accept anything. This is convenient because it allows the `Accept` header to be set to that of the Apache Isis profile for all resources, rather than simply the handful of resources that supported that profile.

6.2. Deprecated

There are also a number of configuration properties that can be used to suppress or simplify the default RO-spec representations.

These configuration properties pre-date the support, introduced in `1.11.0`, for the Apache Isis profile, and are limited by the fact that they are global configuration settings, so cannot be influenced on a request-by-request basis (as is the case with the `Accept` header used for the Apache Isis profile). They have therefore been deprecated, and may be removed in the future.

Nevertheless, those configuration properties are:

Table 2. Deprecated Configuration Properties

Property	**Value** (*default value*)	**Description**
`isis.viewer.restfulobjects.suppressDescribedByLinks`	`true`,`false` (`false`)	Suppresses the "describedby" links (on all representations)

Property	Value (*default value*)	Description
isis.viewer.restfulobjects.suppressUpdateLink	true,false (false)	suppresses the "update" link (on object representation)
isis.viewer.restfulobjects.suppressMemberId	true,false (false)	suppresses the "id" json-prop for object members (on object representation and member detail representations)
isis.viewer.restfulobjects.suppressMemberLinks	true,false (false)	suppresses the "links" json-prop for object members (on the object representation and member detail representations)
isis.viewer.restfulobjects.suppressMemberExtensions	true,false (false)	suppresses the "extensions" json-prop for object members (on the object representation and member detail representations)
isis.viewer.restfulobjects.suppressMemberDisabledReason	true,false (false)	suppresses the "disabledReason" json-prop for object members (on the object representation and member detail representations)
isis.viewer.restfulobjects.objectPropertyValuesOnly	true,false (false)	See discussion below.

For example, these configuration properties could all be added in the WEB-INF/viewer_restfulobjects.properties):

```
isis.viewer.restfulobjects.suppressDescribedByLinks=true
isis.viewer.restfulobjects.suppressUpdateLink=true
isis.viewer.restfulobjects.suppressMemberId=true
isis.viewer.restfulobjects.suppressMemberLinks=true
isis.viewer.restfulobjects.suppressMemberExtensions=true
isis.viewer.restfulobjects.suppressMemberDisabledReason=true
```

If these configuration settings are set in conjunction with using the Apache Isis profile, then the special $$ro property in the representations with reflect these settings.

If the objectPropertyValuesOnly configuration property is set:

```
isis.viewer.restfulobjects.objectPropertyValuesOnly=true
```

then this generates a representation such as:

```
{
    "title" : "Buy milk due by 2014-10-27",
    "domainType" : "TODO",
    "instanceId" : "0",
    "members" : {
        "description" : "Buy milk",
        "category" : "Domestic",
        "subcategory" : "Shopping",
        "complete" : false,
        "versionSequence" : 1,
        "relativePriority" : 2,
        "dueBy" : "2014-10-27",
        "cost" : "0.75",
        "notes" : null,
        "attachment" : null,
        "doc" : null
    },
    "links" : [
        {
            "rel" : "self",
            "href" : "http://localhost:8080/restful/objects/TODO/0",
            "method" : "GET",
            "type" : "application/json;profile=\"urn:org.restfulobjects:repr-
types/object\"",
            "title" : "Buy milk due by 2014-10-27"
        },
        {
            "rel" : "describedby",
            "href" : "http://localhost:8080/restful/domain-types/TODO",
            "method" : "GET",
            "type" : "application/json;profile=\"urn:org.restfulobjects:repr-
types/domain-type\""
        }
    ],
    "extensions" : {
        "oid" : "TODO:0"
    },
}
```

Chapter 7. Hints and Tips

This chapter provides some solutions for problems we've encountered ourselves or have been raised on the Apache Isis mailing lists.

See also hints-n-tips chapters in the:

- the Developers' guide
- the Wicket viewer guide
- the Restful Objects viewer guide (this chapter)
- the Datanucleus ObjectStore guide
- the Security guide
- the Beyond the Basics guide.

Since the Restful Objects viewer is designed for computer programs to interact with (rather than human beings), it can be a little difficult to explore and generally "grok" how it works.

This section provides a few hints-and-tips to help you on your way.

7.1. Using Chrome Dev Tools

This screencast shows how to explore the Restful API using Chrome plugins/extensions, and how we use them to write end-2-end (TCK) tests for the Restful Objects viewer.

7.2. Angular Tips

The hypermedia API exposed by Apache Isis' Restful Objects viewer is intended be support both bespoke custom-written viewers as well as generic viewers. Indeed, we expect most clients consuming the API will be bespoke, not generic.

This page captures one or two tips on using Angular to write such a bespoke client.

7.2.1. Invoking a GET link (eg invoking a query action)

Suppose you have a `CustomerService` providing a `findCustomer` action:

```
public class CustomerService {
    public String id() { return "customers"; }
    @Action(semantics=SemanticsOf.SAFE)
    public Customer findCustomer(
            @ParameterLayout(named="customerName")
            final String customerName) {
        ...
    }
}
```

Restful Objects will expose this as action with the following link that looks something like:

```
{
  "rel" : "urn:org.restfulobjects:rels/invoke",
  "href" :
"http://localhost:8080/restful/services/customers/actions/findCustomer/invoke",
  "method" : "GET",
  "type" : "application/json;profile=\"urn:org.restfulobjects:repr-types/action-
result\"",
  "arguments" : {
    "customerName" : {
      "value" : null
    }
  }
}
```

You can then invoke this using Angular' `$resource` service as follows.

```
var findCustomer = $resource(
"http://localhost:8080/restful/services/customers/actions/findCustomer/invoke?:queryString");
var findCustomerArgs = {
  "customerName": {
      "value": "Fred"
    }
};
findCustomer.get({queryString: JSON.stringify(findCustomerArgs)}, function(data) { ...
} )
```

Here the `:queryString` placeholder in the initial `$resource` constructor is expanded with a stringified version of the JSON object representing the args. Note how the `findCustomerArgs` is the same as the `"arguments"` attribute in the original link (with a value provided instead of `null`).

7.2.2. Invoking a PUT or POST link

If the method is a PUT or a POST, then no `:queryString` placeholder is required in the URL, and the args are instead part of the body.

Use `$resource.put(…)` or `$resource.post(…)` instead.

7.3. Pretty printing

The JSON representations generated by the Restful Objects viewer are in compact form if the deployment type is SERVER (ie production), but will automatically be "pretty printed" (in other words indented) if the deployment type is PROTOTYPE.

7.4. How parse images in RO viewer?

From this thread on the Apache Isis users mailing list:

- *I am trying to display an image in a JavaScript client app, the image comes from an Isis RO web service as a string, but it won't show. Is there something I should do to change the message?*

The RO viewer returns the image as a string, in the form:

```
"Tacos.jpg:image/jpeg:/9j//4AAQSkZJRgABAQEAlgCWAAD/  ...."
```

This is in the form:

```
(filename):(mime type):(binary data in base64)
```

This is basically the `Blob` value type, in string form.

To use, split the parts then format the mime type and base64 data correctly before using as source in an `<img>` tag.

7.5. View Model as Parameter

As discussed on the mailing list.

7.5.1. Query

I must provide a REST service accepting more complex view model as input parameter.

My view model parameter would look like

```
@DomainObject(
    nature = Nature.VIEW_MODEL,
    objectType = "OfferTemplateFilter"
)
@XmlRootElement(name = "OfferTemplateFilter")
@XmlAccessorType(XmlAccessType.FIELD)
@Getter @Setter
public class OfferTemplateFilter {
    public List<String> selectedDeviceManufacturer = new ArrayList<>();
    public List<String> selectedDeviceSizes = new ArrayList<>();
}
```

My REST domain service would be someting like

```
@DomainService(
    nature = NatureOfService.VIEW_REST_ONLY,
    objectType = "OfferRestService"
)
public class OfferRestService {

    @Action(semantics = SemanticsOf.IDEMPOTENT)
    public OfferTemplateSelectorForCustomer
        offerSelectorForCustomer(
            final String subscriberNumber,
            final OfferTemplateFilter filter) {
        return offerSelectorRepository.create(subscriberNumber, filter);
    }
    ...
}
```

I'm wondering how this could be achieved without custom rest service. Ideally the service consumer would post a kind of JSON structure where my view model OfferTemplateFilter would be created?

7.5.2. Possible Answer...

Rather than try to "upload" the `OfferTemplateFilter` view model as a parameter, instead treat it as a resource.

That is:

- have a new service to create an instance of the filter, and then
- update this filter (adding/removing from its two collections).
- When done, pass a reference to the filter to the original REST service, as a regular reference.

Obviously the URL passed in the last step will be rather long and messy, but that's not a problem per-se.

https://lists.apache.org/thread.html/cbd18320bbf6e5c5e767283f9e675cf56e7f4692c109e1e79dbaa90a@%3Cusers.isis.apache.org%3E