

# Restful Objects

Draft v0.69.0

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

The most up-to-date version of this specification may be downloaded from [www.restfulobjects.org](http://www.restfulobjects.org). The site also includes details of known implementations, and other useful information.

Dan Haywood

Licensed under:  
Creative Commons Attribution-ShareAlike 3.0

<http://creativecommons.org/licenses/by-sa/3.0/>





## TABLE OF CONTENTS

A Concepts and Building Blocks.....	A-1
1 Introduction .....	A-3
2 Concepts .....	A-13
3 Optional Capabilities .....	A-47
4 Specified Elements .....	A-55
B Supporting Resources and Representations.....	B-59
5 Home Page Resource & Representation .....	B-61
6 User Resource & Representation .....	B-65
7 Domain Services Resource .....	B-69
8 Version Resource & Representation.....	B-73
9 Objects of Type Resource .....	B-77
10 Error Representation .....	B-81
11 List Representation .....	B-83
12 Scalar Value Representation.....	B-85
C Domain Object Resources & Representations .....	C-87
13 Response Scenarios .....	C-89
14 Domain Object Resource & Representation .....	C-99
15 Domain Service Resource.....	C-113
16 Property Resource & Representation.....	C-114
17 Collection Resource & Representation .....	C-123
18 Action Resource & Representation .....	C-133
19 Action Invoke Resource .....	C-141
D Domain Type Resources.....	D-153
20 Response Scenarios .....	D-155

# Restful Objects

---

21	Domain Types Resource.....	D-159
22	Domain Type Resource .....	D-163
23	Domain Type Property Description Resource .....	D-167
24	Domain Type Collection Description Resource .....	D-171
25	Domain Type Action Description Resource .....	D-175
26	Domain Type Action Parameter Description Resource.....	D-179
27	Domain Type Action Invoke Resource.....	D-183
E Discussions.....		E-191
28	HATEOAS vs Web APIs.....	E-193
29	Personal vs Shared State .....	E-195
30	Dealing with Untrusted Clients .....	E-201
31	Client vs Server Evolution .....	E-203
32	Code Sketch to support Addressable View Models .....	E-206
33	FAQs .....	E-209
34	Ideas for Future Extensions to the Specification .....	E-213

## TABLE OF FIGURES

Figure 1: Resources and Representations .....	A-13
Figure 2: Media Type Layers .....	A-24
Figure 3: Domain Objects vs Domain Types.....	A-51
Figure 4: Home Page Representation.....	B-62
Figure 5: User Representation.....	B-66
Figure 6: Services Representation.....	B-70
Figure 7: Version Representation .....	B-74

## Restful Objects

---

Figure 8: Domain Object Representation .....	C-102
Figure 9: Object Property Representation.....	C-118
Figure 10: Object Collection Representation .....	C-127
Figure 11: Object Action Representation.....	C-134
Figure 12: Action Result for Object.....	C-147
Figure 13: Action Result for List .....	C-149
Figure 14: Action Result for Scalar .....	C-150
Figure 15: Action Result for Void .....	C-152
Figure 16: Domain Type List Representation .....	D-159
Figure 17: Domain Type Representation .....	D-164
Figure 18: Domain Property Collection Representation.....	D-168
Figure 19: Domain Collection Description Representation.....	D-172
Figure 20: Domain Action Description Representation .....	D-176
Figure 21: Domain Action Parameter Description Representation .....	D-180
Figure 22: Domain Type Action Representation.....	D-184



A  
CONCEPTS  
AND  
BUILDING BLOCKS





# 1 INTRODUCTION

Restful Objects is a public specification for a set of RESTful<sup>1</sup> resources<sup>2</sup> by which a client application can interact with a domain model on a server using HTTP. These resources generate JSON representations along with corresponding media types.

This chapter discusses further the goals and approach taken by the spec.

## 1.1 Goals

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.

---

<sup>1</sup> <http://en.wikipedia.org/wiki/REST>

<sup>2</sup> [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

## 1.2 A Uniform Interface

Restful Objects defines a uniform interface to the domain objects. This uniformity is expressed in terms of:

- the format of URLs used to access the domain object resources (though URLs can also be treated as opaque);
- the standard HTTP methods used (GET, POST, PUT, DELETE) to call the resource URLs;
- the standard HTTP headers supported for both request and response;
- the use of standard HTTP status return codes;
- standardized, parameterized media types to fully describe the representation, e.g.:

```
application/json;  
profile="urn:org.restfulobjects:repr-types/object";  
x-ro-domain-type="com.mycompany.myapp.v2.PlaceOrderviewModel"
```

- standard properties within JSON representations;
- a standard representation of links between resources;
- a small number of reserved query parameter names to influence behaviour (e.g. validation).

Existing HTTP standards and supporting W3C standards have been used wherever possible.

## 1.3 Benefits

Because the spec defines a generalized binding for any domain model, it allows the project team to focus on developing the domain model rather than worrying about the intricacies of following a RESTful style. For example, the debate becomes about whether an action is idempotent, not about whether to use HTTP PUT or HTTP POST. And debates about URI structure (should it be customers/{custId}/invoices or should it be invoices/for/{custId}?) disappear because the URL is determined by the responsibilities of the underlying domain objects.

Further specific benefits include:

- **Testing.** Since all business logic is expressed only in domain classes, such business logic may be tested using fast and cheap in-memory unit tests. In contrast, if a RESTful API is custom-written and tuned to specific use cases, then it can only be effectively tested through (slower) integration tests running across a webserver.
- **Documentation.** Restful Objects eliminates the need for the project team to explicitly document the RESTful API to their system, because consumers can infer the structure either directly from the underlying domain classes, or by querying the domain metamodel resources §D.

- **Common server code.** Restful Objects is intended to encourage frameworks to be written that implement the spec (indeed, such frameworks already exist<sup>3</sup>). A project team adopting such a framework can then focus solely on implementing business logic through domain classes, and know that the business logic will be exposed in a standard and consistent fashion.
- **Common client code.** Any generic Restful client written to work with one framework implementation will also inter-operate across any other implementations.

There is further discussion on how the specification enables the RESTful style (along with an FAQ) in §E.

## 1.4 Audience

This document is written for several audiences:

- for developers intending to write a bespoke domain-driven system and want a guidance on how to expose the domain in a RESTful style;
- for developers intending to write a bespoke domain-driven system intended to be hosted on a general-purpose framework that implements this spec;
- for developers intending to write a generic or custom client against a domain object model exposed using Restful Objects;
- for developers intending to write their own general-purpose framework implementation of the specification.

The specification adopts a semi-formal style; providing examples of representations rather than formal grammars. The intention is to make the specification accessible to all its target audiences.

## 1.5 Authorship and License

Restful Objects was conceived by Dan Haywood, who is also the primary author of this document. Substantial contributions have been made by Richard Pawson and Stef Cascarini.

The specification is licensed under Creative Commons Attribution Share-Alike 3.0<sup>4</sup> (the same license as Wikipedia). As such the document may be shared and adapted. However, any derivative work must be attributed back to the author of this document, and must be licensed under the same license to this one.

---

<sup>3</sup> A list of known implementations of the Restful Objects specification is maintained at <http://www.restfulobjects.org>.

<sup>4</sup> <http://creativecommons.org/licenses/by-sa/3.0/>

## 1.6 Style Conventions

Restful Objects defines that URLs, when generated in resources, are absolute. Because the hostname is likely to be a long string and in any case will vary, in the spec it is shown by a '~' placeholder:

```
http://~/
```

Similarly, some URLs also use fully qualified class names. In a Java implementation, this would be a string such as `com.mycompany.myapp.Customer`, while on .NET this might instead be `MyCompany.MyApp.Customer`. For conciseness, this has been shown by an "x" placeholder: `x.Customer`.

There are two important concepts in the spec that both have the name "property": a domain object property, and a property within a JSON representation (the key value of a JSON map). The spec always refers to the former as a "property", and to the latter as a "json-property".

Finally, Restful Objects defines a set of "rel" values for links. To distinguish these from IANA-specified links, these each have the prefix "urn:org.restfulobjects:rels/". This is abbreviated to ".../" in example representations.

## 1.7 Change History

Version	Description
v0.10 ~ v0.31	Internal revisions
v0.40	DH: made ready for wider review
v0.41	Removed change history prior to v0.40; updated table formatting
v0.42	<ul style="list-style-type: none"><li>replaced 403 (forbidden) with 401 (not authorized)</li></ul> Following feedback from REST community: <ul style="list-style-type: none"><li>Additional discussion on the goals of the spec, plus FAQ</li><li>Removed custom X-Representation-Type response header, instead use application/vnd.ro-xxx+json media type (based on Accept header);</li><li>removed other custom request headers (X-Follow-Links, X-Validate-Only, X-Sort-By, X-Page, X-Page-Size) and instead defined standard query params (x-ro-follow-links, x-ro-validate-only etc.)</li><li>added "next" and "previous" links for list representation so that paging is performed in a standard fashion</li><li>changed validation failure to return 400 rather than 412</li><li>specify use of ETags for concurrency control rather than Last-Modified (lack of precision of Last-Modified); returns 412 rather than 409</li><li>changed successful validation to return 204 rather than 304</li></ul>

# Restful Objects

---

Version	Description
V0.43	RP: Review of v42 and minor textual corrections only
v0.44	DH: using "application/json;profile='xxx'" instead of different media types for each representation; add type property for links; reorganized sections 2 and 3; combined parts C and D; added Error representation; clarified that link to action/invoke indicates the media type returned (using profile parameter).
v0.45	DH: Reworked introductory sections in chapter 1.
V0.46	RP: Review and minor edits for clarification and style only
v0.47	DH: Rels and profile parameter now both formally defined as URNs
v0.48	DH: "self", "details" and others are now links rather than json-properties; extended coverage of x-ro-follow-links; moved content from introductory chapter into new "Patterns and Practices" section at end, and expanded its discussion; added diagrams to show links
v0.49	DH: Introduced actionresult representation, with object, list and scalar representations in-lined. Various minor corrections.
v0.50	DH: void actions also now return an actionresult representation.
v0.51	DH: domain type member resources renamed (to "property/collection/action description"); added domain type action resource and representation; object put, property put, property delete, collection put, collection post, collection delete all now return representations (being respectively the update object or member).
v0.52	DH: replaced GET Objects/ resource with GET domainTypes/xxx/typeactions/newTransientInstance/invoke; added isSupertypeOf() type action; added GET domainTypes resource (linked from home page); capabilities now includes an RO spec version; PUT Objects/{oid} and POST Objects/ now accept a cut-down domain object representation, rather than a map of property/values. VersionSpec added to Capabilities representation.
V0.53	RP: Review of recent changes. Suggesting removal of NewTransientInstance from domain type i.e. require it to be done through actions. I18n section added as placeholder.
v0.54	DH: Removed NewTransientInstance; introduced transient domain object representation; added format json-property to simple & formal schemes to handle date/date-time value types; error handling clarifications; regex property changed to 'pattern'
v0.55	RP: Minor edits & corrections, plus comments.

# Restful Objects

Version	Description
v0.56	DH: renamed "capabilities" resource → "version" resource; added parent link to user, services, version, domainTypes resources; removed domainType#newTransientInstance() type action; links in arguments only need specify href; URL encoded argument map passed as entire query string (not as value of an "args" queryparam); more detail on how the Accept header is handled; suppress links[rel=modify] if no properties can be modified; i18n handling described; transient object collections are followed automatically, and may optionally have actions;
V0.57	RP: Removed comments now addressed, Added small number of new comments and minor edits for style/clarity.
v0.58	DH: renamed chapter 3 "Extensions" -> "Optional capabilities"; "x-ro-domainType" as a further parameter for media type, similar to "profile"; added "attachments" (a new optional capability); "format" now also defines "blob" and "clob"; added "links" and "extensions" for action params; "parent" -> "up"; minor edits on resource arg representation; fixed discrepancies between 3.1.1 and detail in representations in part C; reconciled simple schemes vs formal schemes, and move "extensions" json-properties in formal scheme to top-level json-props; "x-ro-invalidReason" json-property introduced.
v0.59	blob/clob property values suppressed from object representation; clarified that client requests blob/clob native representation through the Accept header (URL href can be same as property details); described how blob/clobs are updated; clarified that arguments may also be blob/clobs; extended set of available "attachments" capability values and corresponding values for "x-ro-attachments" reserved query parameter; clarified wording of string properties vs properties whose value is a string; added 'string', 'biginteger' and 'bigdecimal(n)' as new format types, moved section into preceding 'Concepts' changed "members" in object repr and domain description repr to be a map, not a list; changed typeActions in domain description repr to be a map, not a list; changed parameters list in action repr to be a map, not a list; changed parameters list in action description repr to be a map, not a list added "memberOrder" as metadata to both simple and formal schemes; added section in 'patterns & practices' for dealing with untrusted clients;

# Restful Objects

---

Version	Description
v0.60	<p>Added domain object ontology; merged persistent and transient representations of domain objects into single section, and documented links to update/delete properties;</p> <p>removed mention of a application-specific media types (since domainType parameter subsumes this); x-ro-domain-type now must be honoured if specified in Accept header;</p> <p>added code sketch to describe one way that implementations might support addressable view models and conneg;</p> <p>"friendlyName" now considered part of simple metadata, added "pluralForm" as metadata for collections and actions returning lists;</p> <p>added predefined domainType resources to correspond to scalar datatypes (string, int etc) and "list", "set";</p> <p>removed "collectionType" json-property for collection, and combined with "returnType", with "list" and "set" as reserved (platform-independent) type names.</p> <p>property, action and action param link to (formal) or have (simple) returnType json-prop, indicates datatype if scalar, else fully qualified class name;</p> <p>action links to (formal) or has (simple) elementType if returns a collection</p> <p>fixed format of supertype/subtype argument in formal notation;</p> <p>renamed pagination.control to pagination.links;</p> <p>i18n made into an optional capability;</p>
V0.61	RP: Review, edit & comment. No changes to the specification itself.
V0.62	RP: Complete review with edits and comments added.

# Restful Objects

Version	Description
v0.63.0	<p>adopted semantic versioning;</p> <p>rel links:</p> <ul style="list-style-type: none"><li>• rel links now namespaced and with additional qualifier urn:org.restfulobjects:rels/xxx;yyy=zzz ;</li><li>• changed urn:org.restfulobjects/xxx/... → urn:org.restfulobjects:repr-types/xxx (because urn should be urn:xxx:yyy, and to distinguish from rels);</li></ul> <p>blob/clobs are now optional, folded into the optional attachments capability;</p> <p>hyphenation/spelling:</p> <ul style="list-style-type: none"><li>• hyphenated rel names (e.g. 'return-type', not 'returntype');</li><li>• spelling of x-ro-domain-type fixed throughout;</li><li>• changed http://~/domainTypes/... to http://~/domain- types/...;</li><li>• hyphenated the representation types;</li><li>• biginteger &amp; bigdecimal now hyphenated;</li><li>• camelCased all json-props (e.g. resulttype → resultType);</li></ul> <p>using 403 for authentication (not 401);</p> <p>tidied up and correlated together 2.5 scalar types vs 20.3 pre- defined formal types;</p> <p>added chapter 20 summarizing response types for domain types resources (similar to chapter 13 for domain object resources);</p> <p>media types:</p> <ul style="list-style-type: none"><li>• x-ro-domain-type media type parameter usage also described for simple scheme;</li><li>• introduced x-ro-element-type for media type of lists/collections;</li><li>• media type for action results also indicate either the x-ro- domain-type (if returning an object) or x-ro-element-type (if returning an action-result or object-collection) ;</li><li>• renamed "list" media type profile param to "services" (since this is the only time ever returned);</li><li>• removed "scalar" media type profile param since never returned except in-lined within "action-result"</li></ul> <p>POST to Objects/ returns a 201 response code and Location header; renamed "transient" to "proto-persistent" objects</p>



# Restful Objects

Version	Description
v0.64.0	<p>paging, sorting, followLinks, partial arguments and i18n removed as optional capabilities; new chapter to discuss these as possible future enhancements</p> <p>concurrencyChecking promoted to a mandatory capability</p> <p>x-ro-domain-model noted as an optional query string or body parameter for most object resources.</p> <p>the edit in v0.63 to fold of blobs/clobs and attachments together was incomplete – now defined attachments link in properties repr, and how to GET a blobClob value via conneg. Also more info on the 406 response code.</p> <p>changed {oid} to {domainType}/{instanceId} throughout. The "oid" json-property is replaced with "instanceId".</p> <p>changed /Objects to /Objects/{domainType}; no longer any need to provide the links[rel=.../describedBy] property.</p> <p>added "domainType" json-property to domainType repr.</p> <p>in simple scheme, "domainType" json-prop is now merely a "unique identifier" to the type (previously it was described as the fully qualified class name)</p> <p>added table describing domain service resources to §2.3.</p> <p>domain object action and object action invoke chapters now also updated to indicate that they apply to domain services also</p> <p><b>Content-Length</b> header be returned for all representations.</p> <p>in v0.63.0: added Pragma, No-Cache, Expires headers</p> <p>Renamed "NONE", "SHORT" and "LONG" cache placeholders to "TRANSACTIONAL", "USER_INFO" and "NON_EXPIRING"; allow implementations to support reverse proxies for TRANSACTIONAL</p> <p>Removed ETag as a header from any domain type responses</p> <p>201 can be returned from action result §19.3</p>
v0.65.0	RP: minor edits
v0.66.0	<p>added 428 as a new syntax error response code</p> <p>added 422 as a new validation error response code</p> <p>changed profile=".../services" back to profile=".../list" in prep for future enhancement to support ~/objects/{domainType}.</p> <p>tidying up/improving words in the "discussions" chapters</p>
V0.67.0	RP: Thorough review of main text (not diagrams), with edits & comments added. Icon16 and Icon32 extensions deleted.

# Restful Objects

---

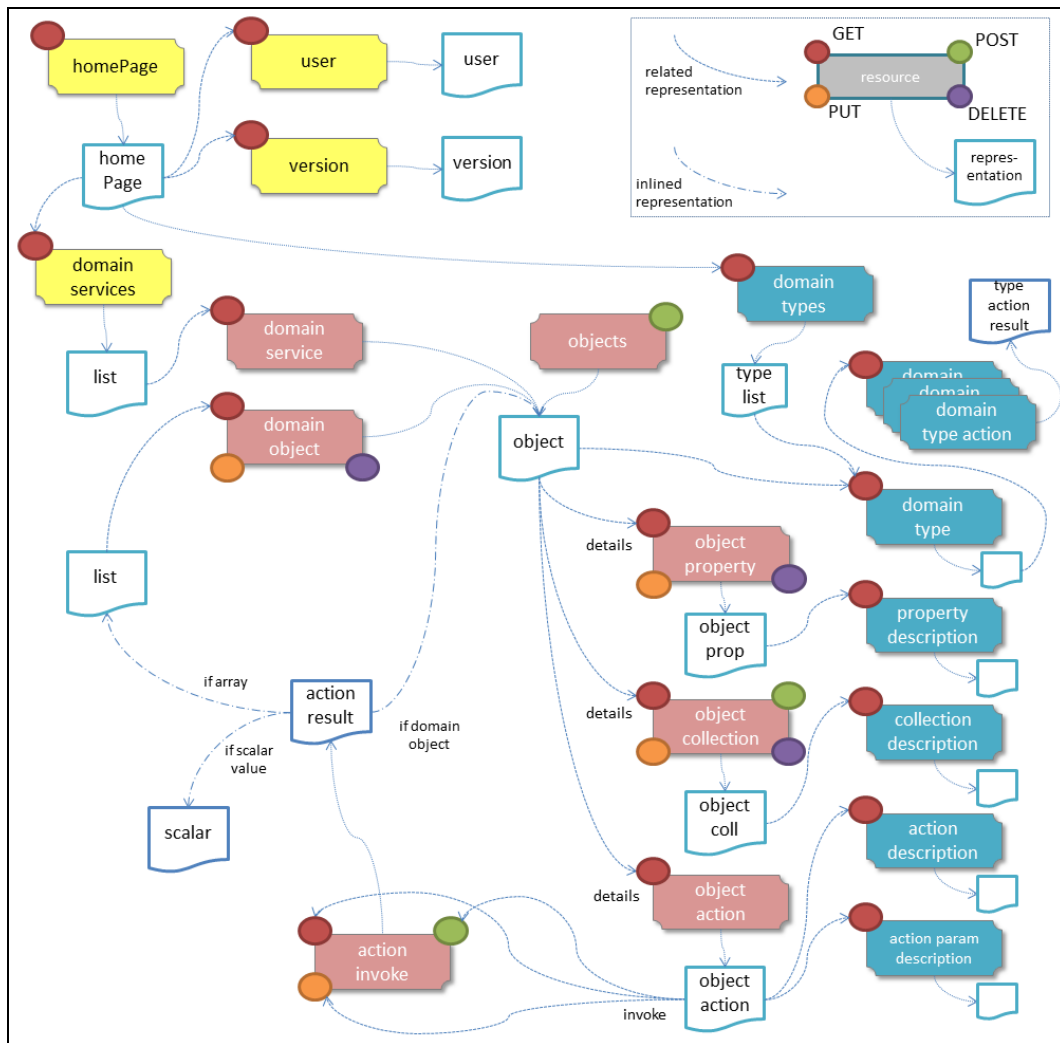
Version	Description
v0.68.0	<p>Addressing RP's comments from 0.67.0; in particular:</p> <ul style="list-style-type: none"><li>• Treatment of number JSON values with no "format" hint changed, now in-line with ECMAScript standard;</li><li>• Moved some stale material on partial arguments to future enhancements section</li><li>• Removed statement that mutating properties or collections should suppress the self link (per RP's comment on 0.67.0)</li></ul> <p>422 as new response code for POST to "Object of Type" resource</p> <p>Clarifications on x-ro-domain-type media type parameter:</p> <ul style="list-style-type: none"><li>• now set to domain type id {domainType}, rather than fully qualified class name,</li><li>• returned only for Content-Type response header, not Accept request header</li><li>• Returned value should be run-time type, not compile-time type</li><li>• No longer include as a parameter within the "type" json-prop of links (because should be the runtime type of the object);</li><li>• "x-ro-element-type" now also set for a "list" as well as "action-result" and "object-collection" repr-type (only one example currently: the services list)</li></ul> <p>(As result of changes to x-ro-domain-type), conneg now a "future enhancement"; updated some FAQs</p> <p>Specify format of Warning header, as per RFC2616</p>
V0.69.0	<p>Several ncorrect references to 'fully-qualified name' changed to 'domain type id'.</p>

## 2 CONCEPTS

This section describes the main concepts that underpin Restful Objects.

### 2.1 Resources vs. Representations

The following diagram shows the RESTful resources defined by the specification, along with the representations that they generate:



**FIGURE 1: RESOURCES AND REPRESENTATIONS**

In this diagram:

- Each shaded box indicates a resource that can be invoked. The colour indicates the part of the spec where the resource is defined; yellow is supporting resources §B, pink is domain object resources §C, blue is domain type resources §D;
- The circles on the corner of each shaded box indicate the HTTP methods (GET, PUT, POST, DELETE) supported by that resource;

Most of the resources show a dotted-arrow pointing to a (white) document icon, to indicate the representation returned by the resource. Some resources return their own, unique, representation (e.g. the home page resource, the user resource). However, some representations (most notably the object representation) may be returned by several different resources.

- Dotted lines from a representation to a resource indicate that the representation contains a link to another resource  
Most of the significant GET links are shown; to reduce visual clutter most PUT, DELETE and PUT links are not shown;
- In-lined representations (of object, lists, or scalars within an action result) are shown by a dash-dot-dash line.

### *Resource Relationships*

The home page resource §B5 is intended to act as a start page: a well-known location from which a client can start to interact with the server. The representation returned by this resource provides links to three further resources:

- The user resource §B6 represents the currently logged-in user; the returned User Representation provides details such as their user name and roles.
- The domain services resource §B7 returns a list representation §B11, which contains a link to each domain service resource §C15. A domain service is a singleton object that has actions but no state. The representation returned by a service resource is an object representation, but with content clearly indicating that the object it represents is a service.
- The version resource §B8 provides a mechanism for the client to discover the version of the spec and of the implementation, as well as querying any optional capabilities of the implementation (e.g. whether it supports direct deletion of persisted objects).

The domain object representation §C14.4 is the most important representation within Restful Objects; from it various sub-resources can be accessed. The most immediate such sub- resources are for:

- each property of the object (§C16.4)
- each collection of the object (§C17.5), and
- each action of the object (§C18.2).

Following these links it is possible to walk the graph to other domain object resources associated via properties or collections, and/or to modify the contents of properties and collections.

An action on an object may be invoked through its 'action invoke' sub-resource (§C19); the HTTP method to use is dependent on the action's semantics. The representation returned by invoking the action will either be a scalar value §B12, a representation of the returned domain object §C14.4, or a list §B11 (linking to multiple domain object resources §C14)

As well as providing access to sub-resources, the domain object resource also allows multiple properties to be updated, and objects to be deleted in a single transaction. (The latter is an optional capability §B8.)

Restful Objects defines two schemes by which implementations may provide domain type information – such as whether a property is mandatory or not – to clients. The "simple" scheme in-lines certain key domain type information directly within the domain object representation. The "formal" scheme, by contrast, defines separate 'domain type' resources and corresponding representations, with the domain object representations linking to these domain type resources. There are six such resources (§D21.2, §D22.2, §D23.2, §D24.2, §D25.2, and §D26.2).

Restful Objects also defines a general mechanism to enable new domain object instances (that have been created by an action) to then be persisted, through the Objects Of Type resource §B9.

## 2.2 Domain Object Ontology

The representations defined by Restful Objects are RESTful in the sense that they provide relevant links to other resources that the client can follow: this is Restful Objects' support for HATEOAS §E28.1. Depending on the nature of the domain object being represented, links may or may not be present:

- to properties §C16.1 and collections §C17.1 of the object
- to actions §C18.1 on the object
- to update all properties §C14.2 of the object
- to persist the object §B9.1
- to delete the object §C14.3

The following sections describe how the different types of domain object will result in the presence or absence of specific links. (Note that in all cases, a link is only ever provided if the client has the correct security permissions for that capability).

### *Persistent domain entity*

The most common type of domain object that Restful Objects deals with is a persistent domain entity: an object instance that exists from one interaction to the next, whose state is stored in a database (for example in an RDBMS table) and which is potentially visible to all clients.

Typically a representation of a persistent domain entity includes links to the entity's state (its properties and collections).

The representation will contain links to the object's actions, by which domain object behaviour can be invoked. This is a key piece of HATEOAS support.

Assuming that at least one property is updatable, the "update properties" link will also be present. And if object deletion §C14.3 is supported by the implementation, then the delete object link will also be present.

The "persist object" link will not be present because this object is already persisted.

Examples of persistent domain entities are Customer, Order, OrderItem, and Product.

### *Proto-persistent domain entity*

A proto-persistent domain entity is an object instance that is created as a result of an interaction and immediately represented back to the client, without having been persisted first. The ultimate persistence of the entity is therefore under the control of the client.

Unlike a persistent domain entity, for a proto-persistent domain entity there is no server-side resource to address after the first interaction which returns its representation. This means that its representation must have all the state (properties and collections) in-lined within the representation. There are no links to update the object's properties, nor to delete that object. And there are no links to any domain object actions. The only link that is available is the one to persist the object.

For example, a Customer object might provide a `createOrder()` action that returns (the representation of) a proto-persistent Order as its result, with certain properties set as required. The client would be expected to complete relevant details for the Order, and then to use the provided `rel=".../persist"` link in order to persist the Order. Thereafter that order will always be handled as a persistent domain entity.

### *View model*

A view model is a type of domain object that projects the state of one or more domain entity instances. This is typically done in support of a particular use case.

View models may also be used to provide a stable layer of abstraction. This is necessary when the deployment cycle for the Restful server and its client(s) are different: the server must ensure that any representations consumed by its client(s) remain backwardly compatible.

An example of a view model might be `OrderHistoryReport`, which aggregates information about a number of historical orders (e.g. so they can be graphed or plotted in some form).

View models are not persisted and so (like proto-persistent entities) their representation includes the state of the view model but no behaviour. However, unlike proto-persistent entities, they provide no persist link. In fact, such representations contain no links at all.

### *Addressable view model*

Because simple view models provide no links, they leave the consuming client at a dead-end; in order to do further work the client must use information saved from a previous representation. In other words, the HATEOAS principle is broken.

In order for any action link to work, the object must have some notion of identity from one interaction to the next. Where a view model instance does have such an identity we describe it as an 'Addressable View Model'.

How this identity is managed is implementation-specific, but typically an addressable view model will be closely associated with an underlying persistent domain entity (by convention or some other means); the implementation can then use that underlying entity in order to re-create some server-state. See §E32.1 for a code sketch as to how this might be accomplished.

In theory addressable view models could also provide links to related properties or collections. However, because the purpose of a view model is also to expose a stable set of state for a particular use case, implementations are more than likely expected to simply in-line the property or collection values in their representation.

A good example of an addressable view model is Order/OrderItems, where a single representation has the state of a (persistent) Order along with all its associated OrderItems. However, such a view model would also provide actions that could delegate to the underlying Order object.

### *Domain service*

The last category of domain objects is a domain service. A domain service is a singleton domain object that acts as a repository for locating existing domain entities, as a factory for creating new entities, or provides other services to domain objects.

Domain services typically do not have state (properties or collections), only behaviour (actions). They also cannot be updated, persisted or deleted.

## 2.2.1 Summary

The following table summarizes the links (to other representations) that may be present in the object representation §C14.

	property	collection	action	persist	update properties	delete
Persistent entity	yes	yes	yes	---	yes	yes
Proto-persistent entity	--- (in-lined)	--- (in-lined)	---	yes	---	---
View model	--- (in-lined)	--- (in-lined)	---	---	---	---
Addressable view model	--- (in-lined)	--- (in-lined)	yes	---	---	---
Domain service	---	---	yes	---	---	---

In the above table "yes" indicates that a link to that other resource may be present; "in-lined" means that the value of the property/collection is part of the object representation itself (as one large JSON document).

As noted above, it isn't strictly necessary to distinguish these different types of domain objects; a client can only follow the links that it is provided in the representation. However, where there is likely variation in the presence or not of a link, the spec uses the above terms as a way to explain why that variation occurs.



## 2.3 Domain Object & Service Resources

The following table summarises the resources that relate directly to domain objects.

	Objects Of Type §B9	Object §C14	Object Property §C14.4	Object Collection §C16.4	Object Action §C17.5	Object Action Invoke §C18.2
	Objects/ {DType}	Objects/ {DType}/ {IID}	Objects/ {DType}/ {IID}/ Properties/ {Property}	Objects/ {DType}/ {IID}/ Collections/ {Collection}	Objects/ {DType}/ {IID}/ Actions/ {Action}	Objects/ {DType}/ {IID}/ Actions/ {Action}/ invoke
GET	n/a – 405	object summary, member summary, property values	property details and value	collection details and content	action prompt	invoke (action known to be query-only)
PUT	n/a – 405	update or clear multiple property values	update or clear value	add object (if Set semantics)	n/a – 405	invoke (action known to be idempotent)
DELETE	n/a – 405	delete object	clear value	remove object	n/a – 405	n/a – 405
POST	persist instance	n/a – 405	n/a - 405	add object (if List semantics)	n/a – 405	invoke (action not known to be idempotent)

The columns indicate the domain object resources shown in the Figure 1, plus the Objects Of Type resource §B9 used for persisting new object instances.

The header row indicates the resources as templated URIs<sup>5</sup>:

- {DType} is the *domain type identifier* that uniquely represents the *domain type*. Depending on the implementation this may take an abbreviated form e.g. "CUS" for Customer, or could be the fully qualified class name, eg "com.mycompany.myapp.Customer". The spec requires only that the value is unique;
- {IID} is the *instance identifier* that uniquely identifies an object instance within its type: e.g. "123" for customer with id=123;

<sup>5</sup> <http://tools.ietf.org/html/draft-gregorio-uritemplate-08>

## Restful Objects

---

- {Property}, {Collection} and {Action} are unique identifiers for a property, collection or action of the object, e.g. "firstName", "orders", or "placeOrder"

For brevity, the combination of domain type/instance identifier {DType}/{IID} is also termed the object identifier, or oid.

The body of the table indicates which HTTP methods may be used to access these resources.

The HTTP GET method is the most widely supported across the various resources, and is used to obtain a summary representation of an object §C14.4 (e.g. a Customer instance), or detailed information about a specific property of an object §C16.4 (e.g. Customer.firstName) or about a specific collection §C17.5 (e.g. Customer.orders).

In addition, HTTP GET is used to obtain a representation of an object action §C18.2, such as the Customer's placeOrder() action. Getting the representation of an action does *not* invoke the action; rather the returned representation *describes* the action, providing such information as the arguments and the HTTP method required to invoke the action.

Modifying the state of a domain object is performed through resources supporting HTTP PUT, DELETE or POST. The HTTP method to use to request the modification depends upon the resource's semantics:

- if the resource being called is idempotent, meaning that it *will* change persisted objects but calling that same resource again (with the same inputs) will have no further effect<sup>6</sup>, then either HTTP PUT or HTTP DELETE is used
- if the resource being called is not idempotent, then HTTP POST is used

Whether HTTP PUT or DELETE is used depends on context: if a new data value is being provided then PUT is used, if a value is being cleared or data removed in some way then DELETE is used.

So, properties can be set to a new value using HTTP PUT §C16.2, or can be set to null using HTTP DELETE §C16.3. Modifying multiple properties is accomplished using an HTTP PUT to the object resource §C14.2.

---

<sup>6</sup> In computer science terms, an idempotent function is one that if repeatedly applied, has the same impact as being applied once ; see <http://en.wikipedia.org/wiki/Idempotence>.

## Restful Objects

---

For collections things are a little more involved because the HTTP method to use depends upon the collection's semantics. The most common situation is where the collection follows 'Set' semantics (in other words, it does not allow duplicates to be added). In this case the HTTP PUT §C17.2 is used; if the object exists then the request to add it is ignored, so this is idempotent. If the collection does allow duplicates (in other words, it follows 'List' semantics) then HTTP POST §C17.3 is used. In either case references are removed from the collection using HTTP DELETE §C17.4.

Actions are invoked through the '/invoke' sub-resource. The method used depends on the action's semantics: if the action is idempotent, then PUT §C19.2 is used, otherwise POST §C19.3 is used. However, there is a further special case for actions: if the action is query-only and so makes no changes to persisted objects at all<sup>7</sup>, then Restful Objects allows HTTP GET §C19.1 to be used to invoke the action.

Whether an action is query-only or is idempotent is down to the implementation to determine and to enforce.

Not every HTTP method applies to every resource, and where it does not the specification requires that a 405 ('method not allowed') status code is returned. This will be accompanied by an **Allow** header to indicate which methods are allowed by the resource<sup>8</sup>. A 405 will also be returned if the client attempts, for example, to invoke an action with a GET that is not query-only (or cannot be determined to be so by the server implementation).

In addition to the domain object resources, there are also resources for domain services. However, domain services have no state, so there are no subresources for properties or collections:

---

<sup>7</sup> In computer science terms, a query-only action is "side-effect-free": it does not make any change to persisted data. See [http://en.wikipedia.org/wiki/Side\\_effect\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Side_effect_(computer_science)). A query only action is always idempotent)

<sup>8</sup> <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.7>

## Restful Objects

	Service §C15	Service Action §C17.5	Service Action Invoke §C18.2
	Services/{ServiceId}	Services/{ServiceId}/ Actions/{Action}	Services/{ServiceId}/ Actions/{Action} /invoke
GET	service summary, action summary	action prompt	invoke (action known to be query-only)
PUT	n/a – 405	n/a – 405	invoke (action known to be idempotent)
DELETE	n/a – 405	n/a – 405	n/a – 405
POST	n/a – 405	n/a – 405	invoke (action not known to be idempotent)

The services/{serviceId} URL is broadly equivalent to objects/{domainType}/{instanceId}. However PUT and DELETE are not supported (because domain services have no properties and cannot be deleted).

The services/{serviceId}/actions/... subresources are directly equivalent to objects/{domainType}/{instanceId}/actions/... subresources, and support the exact same HTTP methods.

### 2.3.1 Example Resource URLs

The following table lists some example URLs for accessing resources:

Resource Type	Resource
object	<a href="http://~/objects/ORD/123">http://~/objects/ORD/123</a>
property	<a href="http://~/objects/ORD/123/properties/createdOn">http://~/objects/ORD/123/properties/createdOn</a>
collection	<a href="http://~/objects/ORD/123/collections/items">http://~/objects/ORD/123/collections/items</a>
action	<a href="http://~/objects/ORD/123/actions/placeOrder">http://~/objects/ORD/123/actions/placeOrder</a>
action invocation	<a href="http://~/objects/ORD/123/actions/placeOrder/invoke">http://~/objects/ORD/123/actions/placeOrder/invoke</a>
service	<a href="http://~/services/x.CustomerRepository">http://~/services/x.CustomerRepository</a>

In the example URLs the "ORD" is the domain type identifier, while the "123" is the instance identifier. Together these identify a persisted instance of a domain object of a particular type (an Order, in this case). The format of both the domain type identifier and the instance identifier is implementation-specific, though both must be URL-encoded. (For security reasons, the instance identifier may even be encrypted – see §E30.)

# Restful Objects

## 2.3.2 Example usage scenario

The following table shows an example of the interactions between a client application and a Restful Objects server, for a simple web-shopping scenario. It is rendered as a sequence of HTTP calls.

Description	Method	URL	Request Body	Returned representation
Go to the home resource	GET	http://~/	-	Home Page
Follow link to list of Services available	GET	http://~/services	-	List (of links to Services)
Follow link to the ProductRepository service	GET	http://~/services/x.ProductRepository	-	Object (representing a Service)
Follow link to 'Find By Name' action	GET	http://~/services/x.ProductRepository/actions/FindByName	-	Action (to display to user as a dialog)
Invoke this (query-only) action with "cycle" as the parameter	GET	http://~/services/x.ProductRepository/actions/FindByName/invoke/?Name=cycle	-	Action result in-lining list of links to Product objects
Follow the link to one of the Product objects in the collection	GET	http://~/objects/object/x.Product/8071	-	Object of type Product
Invoke the (zero parameter) action 'AddToBasket' on this object	POST	http://~/objects/object/x.Product/1234/actions/AddToBasket/invoke	-	-
Invoke the action 'ViewBasket...' on the BasketService	GET	http://~/services/x.BasketService/actions/ViewBasketForCurrentUser/invoke	-	Action result in-lining list of links to Item objects
Modify the Quantity property on the item just added	PUT	http://~/objects/object/x.Item/1234/properties/Quantity	Property representation with value=3	-
Delete a (previously added) item from the Basket	DELETE	http://~/objects/ x.Item/55023	-	-

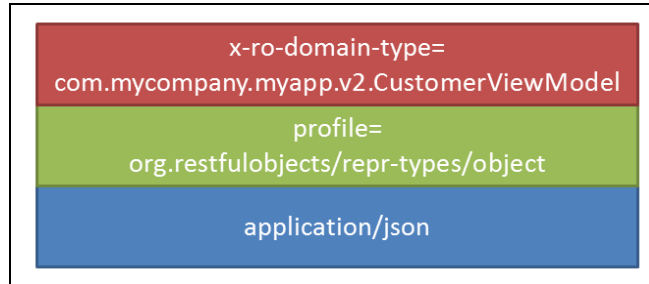
## 2.4 Media Types (Accept and Content-Type)

Web browsers typically use the media type in order to determine how to render some returned content. For example, *text/html* indicates an HTML page, while *image/png* and *image/svg* are different types of images.

# Restful Objects

Rather than defining its own set of custom media types, the specification uses the standard media type for JSON representations, *application/json*, and then uses media type parameters that indicate the structure and semantics of the JSON.

Depending on the representation, there are additional parameters: "profile" and either "x-ro-domain-type" or "x-ro-element-type":



**FIGURE 2: MEDIA TYPE LAYERS**

As the diagram shows, the "profile" parameter refines the semantics of *application/json*, and the "x-ro-domain-type" parameter refines the semantics of "profile" parameter of object representations. The "x-ro-element-type" parameter similarly refines the semantics of "profile" for list/collection representations.

Note that the spec also supports non-JSON media types, such as *application/pdf* and *image/jpeg*, for blobs and clobs. See §3.3.

## 2.4.1 RepresentationType ("profile" parameter)

The *representation type* is used to indicate the nature of the representation, and is specified as the value of the "profile" parameter<sup>9</sup>. By inspecting the value, the client can dynamically determine how to deal with a representation.

The format of the media type with representation type is therefore:

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

Every representation defined by the Restful Objects spec has a corresponding representation type:

Representation type	Indicates a representation of
homepage	the start page §B5
user	the user requesting the resource §B6
version	the version of the spec and implementation §B8

<sup>9</sup> <http://buzzword.org.uk/2009/draft-inkster-profile-parameter-00.html>

Representation type	Indicates a representation of
list	a list of references to domain services or objects §B11
object	a domain object instance (or a service, which is a singleton object) §C14.4
object-property	a domain object property §C16.4
object-collection	a domain object collection §C17.5
object-action	a domain object action §C18.2
action-result	result of invoking a domain object action §C19.4
type-list	a list of domain types §D21.2
domain-type	a domain type §D22.2
property-description	a domain property's description §D23.2
collection-description	a domain collection's description §D24.2
action-description	a domain action's description. §D25.2
action-param-description	an action parameter's description §D26.2
type-action-result	result of invoking a domain type action §D27.
error	An error was generated, §B10.

## 2.4.2 Domain Type ("x-ro-domain-type" parameter) and Element Type ("x-ro-element-type" parameter)

While the "profile" parameter informs the client of the representation type, in the case of an object representation (that is, for profile="urn:org.restfulobjects:repr-types/object") there is no easy way for the client to distinguish between, for example, (the representation of) a Customer and (the representation of) an Order.

For clients that want to handle such representations differently, the spec defines an additional **"x-ro-domain-type"** parameter<sup>10</sup>.

Similarly, when a list of objects is returned (that is, for "profile" is any of "urn:org.restfulobjects:repr-types/action-result", "urn:org.restfulobjects:repr-types/object-collection" or "urn:org.restfulobjects:repr-types/list" ), there is no easy way for the client to know what type the elements of the list are.

Therefore, the spec defines an additional **"x-ro-element-type"** parameter.

---

<sup>10</sup> Unlike the "profile" parameter, there is no standard or semi-standard parameter to reuse. The "x-ro-" prefix of the "x-ro-domain-type" parameter is to avoid name clashes with other emerging standards.

## Restful Objects

---

The value of both of these parameters is a domain type identifier {domainTypeId}. For "x-ro-domain-type" the value should be of the actual runtime type, for "x-ro-element-type" it should be of the collection's compile-time type.

For example, the media type for the representation of a Customer might be:

```
application/json;
profile="urn:org.restfulobjects:repr-types/object";
x-ro-domain-type="CUS"
```

while the representation of a collection of Customers might be:

```
application/json;
profile="urn:org.restfulobjects:repr-types/object-collection";
x-ro-element-type="CUS"
```

where in both cases "CUS" is the domain type identifier for this Customer class.

In the case of a view model, the "x-ro-domain-type" value would more likely include a version number, eg:

```
application/json;
profile="urn:org.restfulobjects:repr-types/object";
x-ro-domain-type="OHVM2"
```

where, say, "OHVM2" is the unique domain type id corresponding to the class `com.mycompany.myapp.viewmodels.v2.OrderHistory`.

The "x-ro-domain-type" and "x-ro-element-type" parameters are also returned for action result representations which wrap a domain object or a list of domain objects.

For example, an action that returned a single Customer would return a media type (under the simple scheme) of:

```
application/json;
profile="urn:org.restfulobjects:repr-types/action-result";
x-ro-domain-type="CUS"
```

while an action that returned a list of Customers (under the simple scheme) would be:

```
application/json;
profile="urn:org.restfulobjects:repr-types/action-result";
x-ro-element-type="CUS"
```

In all the above cases the client can use this value to process the representation accordingly; for example, rendering it with a different view template.



---

## 2.4.3 Handling of Accept headers

---

The HTTP protocol<sup>11</sup> defines the **Accept** request header for the client to specify which media types it can consume; the server then indicates the actual media type using the **Content-Type** response header. If the server is unable to return the requested type, then it must return a 406 "not acceptable" status return code.

Restful Objects defines the following behaviour:

- if the client provides no **Accept** header, then the server may serve up a representation of any content type
- if the client provides an **Accept** header of \*/\*, or application/\*, then any representation may be returned. In this case any "profile" parameter will be ignored
- if the client specifies one or more "profile" parameters, then the server must ensure that the returned representation is one of those that is acceptable. If it is not, then a 406 must be returned.

Note however that if the client specifies the "x-ro-domain-type" parameter, then this is ignored by the server. This means that the client cannot currently use this parameter to ensure that, for example, v1 of a view model is returned rather than v2. Support for content negotiation through the "x-ro-domain-type" parameter in this way is likely to be introduced in a future version of the spec, see §E34.1.

If the client does elect to specify "profile" parameters, then it should take care to always include the error profile. In other words, a request that is expected to return a domain object representation should provide an **Accept** header of:

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

If the error profile is omitted and a (server-side) error occurs, the server may still return the error representation, but must return a 406 (rather than the usual 500 error).

---

## 2.4.4 Browsing the RESTful API

---

During development it can be helpful to browse a RESTful API directly, using a browser plugin such as [RESTConsole](#) or [JSONView](#). Such plugins provide such features as folding of the JSON representation, and

---

<sup>11</sup> <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>

automatic detection of links in the representation so that they can be followed (with a GET).

Although designed to consume JSON, some of these tools incorrectly set the **Accept** header to a value other than *application/json*. Normally, this would result in a 406 ("Not acceptable") response error. In order to accommodate the use of such tools, implementations may wish to provide a "non-strict" mode of operation to suppress **Accept** header validation. However, this is not part of the spec.

Even if **Accept** header validation has been suppressed, the **Content-Type** returned should be set to *application/json* along with the "profile" (and any other) parameter.

## 2.5 Scalar datatypes and formats

JSON defines only the following scalar datatypes<sup>12</sup>:

- Number (double precision floating-point format)
- String (double-quoted Unicode, UTF-8 by default)
- Boolean (true or false)

The JSON schema specification<sup>13</sup> also defines:

- Integer (a number with no floating-point value)

Most notably, JSON does *not* define a native datatype to represent date, time or date/time. Also, it does not define datatypes to represent arbitrarily accurate decimal or integer numbers. Therefore, representing values of these datatypes requires that the information be encoded in some way within a JSON string value.

The Restful Objects spec defines the "**format**" json-property as an additional modifier to describe how to interpret the value of a string or number json-property.

The values of the "**format**" json-property for string values are<sup>14</sup>:

- string
  - The value should simply be interpreted as a string. This is also the default if the "**format**" json-property is omitted (or if no domain metadata is available)

---

<sup>12</sup> <http://json.org/>, also [http://en.wikipedia.org/wiki/JSON#Data\\_types.2C\\_syntax\\_and\\_example](http://en.wikipedia.org/wiki/JSON#Data_types.2C_syntax_and_example)

<sup>13</sup> <http://tools.ietf.org/html/draft-zyp-json-schema-03>, section 5.1.

<sup>14</sup> A number of these are also defined in the JSON schema, <http://tools.ietf.org/html/draft-zyp-json-schema-03>, section 5.23

- date-time
  - A date in ISO 8601 format of YYYY-MM-DDThh:mm:ssZ in UTC time.
- date
  - A date in the format of YYYY-MM-DD.
- time
  - A time in the format of hh:mm:ss.
- utc-millisec
  - The difference, measured in milliseconds, between the specified time and midnight, 00:00 of January 1, 1970 UTC.
- big-integer(n)
  - The value should be parsed as an integer, scale n.
- big-decimal(s,p)
  - The value should be parsed as a big decimal, scale n, precision p.
- blob
  - "binary large object": the string is a base-64 encoded sequence of bytes.
- clob
  - "character large object": the string is a large array of characters, for example an HTML resource

The values of the "**format**" json-property for number values are:

- decimal
  - the number should be interpreted as a float-point decimal.
- int
  - the number should be interpreted as an integer.

If there is no "format" json-property or domain metadata, then the value is interpreted according to standard Javascript rules, as documented in the EcmaScript standard<sup>15</sup>. In essence: if there is NO decimal point and the number is in the range [-9,007,199,254,740,992, +9,007,199,254,740,992], then it is an integer. Otherwise, the number is a 64-bit IEEE754 floating point number.

Note that the internationalization of dates (e.g. formatting a date as MM/DD/YYYY for the en\_US locale) is a responsibility of the client, not the server implementation. Dates should always be provided in the formats described above; the *Accept-Language* header should be ignored.

If the implementation supports the formal metamodel scheme §3.1.2, then each of these datatypes has a corresponding pre-defined domain type resource §D21.3.

Support for blobs and clobs is an optional capability, and is discussed further in §3.3.

---

<sup>15</sup> <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>

## 2.6 Values

The spec defines JSON representations for the values of object properties, collection references and argument values. These either being of a value type (e.g. String, date, int) or a reference type (e.g. a link to a Customer, OrderStatus). This is true both for property values and for argument values; collections only ever contain reference types.

For value types, the value that appears in the JSON is the actual JSON value, either a number, a Boolean, a string or a null. In the case of a string value this may be the formatted version of some other datatype, such as a date §2.5.

For example, if the 'createdOn' property is a date, then its value would be represented thus:

```
"createdOn": {  
  ...  
  "memberType": "property",  
  "value": "2011-06-14",  
  "format": "date",  
  ...  
}
```

For reference properties, the value held is a link. For example, if 'orderStatus' is a property of type OrderStatus, then its representation would be something like:

```
"orderStatus": {  
  ...  
  "memberType": "property",  
  "value": {  
    "rel": ".../value;property=\"orderStatus\"",  
    "href": "http://~/objects/ORS/IN_PROGRESS",  
    "type": "application/json;profile=\".../object\"",  
    "title": "In Progress",  
    "method": "GET"  
  },  
  ...  
}
```

## 2.7 Link representation

Every JSON representation may have relationships to other representations, and each such relationship is described through a standard **link** representation with the format:

```
{  
  "rel": ".../xxx",  
  "href": "http://~/objects/ORD/123",  
  "type": "application/json;profile=\".../object\"",  
  "method": "GET",  
  "title": "xxx",  
  "arguments": { ... },  
  "value": { ... }  
}
```

# Restful Objects

where:

JSON-Property	Description
<b>rel</b>	Indicates the nature of the relationship of the related resource to the resource that generated this representation; described in more detail below
<b>href</b>	The (absolute) address of the related resource. Any characters that are invalid in URLs must be URL encoded.
<b>type</b>	The media type that the linked resource will return; see §2.4.
<b>method</b>	The HTTP method to use to traverse the link (GET, POST, PUT or DELETE)
<b>title</b>	(optional) string that the consuming application may use to render the link without having to traverse the link in advance
<b>arguments</b>	(optional) map that may be used as the basis for any data (arguments or properties) required to follow the link. Discussed further below.
<b>value</b>	(optional) value that results from traversing the link. This is to support eager loading of links by resources. For example, an Order representation may have a collection of OrderItems, and may want to provide that representation to avoid an additional round-trip request by the client.

## 2.7.1 "rel"

The **"rel"** json-property indicates the nature of the relationship of the related resource to the resource that generated this representation. The value of this property is a URN, meaning that it is unique value within a defined namespace (specific to Restful Objects).

The value of the **"rel"** json-property either takes one of the IANA-specified rel values<sup>16</sup> or a value specific to Restful Objects.

### 2.7.1.1 IANA-specified rel values

rel	Description
<b>describedby</b>	"Refers to a resource providing information about the link's context"; in other words the domain metamodel information about a domain object or object member
<b>help</b>	"Refers to context-sensitive help"
<b>icon</b>	"Refers to an icon representing the link's context." A scalable icon for any purpose
<b>previous</b>	"Refers to the previous resource in an ordered series of resources"

<sup>16</sup> <http://www.iana.org/assignments/link-relations/link-relations.xml>

## Restful Objects

rel	Description
next	"Indicates that the link's context is a part of a series, and that the next in the series is the link target".
self	"Conveys an identifier for the link's context", in other words, following this link returns the same representation. Discussed further in §2.8.
up	Link from member to parent object/type, or from action param to its action

### 2.7.1.2 Restful Objects-specified rel values

The format of Restful Objects-specified rel values is:

```
urn:org.restfulobjects:rels/xxx[;yyy=zzz;www=vvv]
```

where

- urn:org.restfulobjects:rels/
  - is a fixed prefix indicating that the rel is defined by the Restful Objects specification
- xxx
  - is a unique value for the rel within the above namespace
- yyy=zzz, www=vvv
  - are additional parameters that are used for some rel values to disambiguate the link

The optional parameters are modelled after the optional parameters of media types (§2.4.1, §2.4.2). Using them clients can, for example, distinguish a link more precisely without having to rely on the location of the link within the JSON representation.

For example:

```
urn:org.restfulobjects:rels/details;property=\"deliveryoption\"
```

is the rel value of a link to property details resource, §C16.1.

The table below lists all the supported rel values defined by Restful Objects. For brevity the "urn:org.restfulobjects:rels/" prefix is abbreviated to ".../".

rel	Parameters	Description
.../action		Description of an action §D25, as linked from a domain type §D22
.../action-param		Description of an action parameter §D26, as linked from an action resource §D25
.../add-to;	collection=\"collectionName\"	Add to a domain object collection §C17.2, §C17.3

# Restful Objects

rel	Parameters	Description
.../attachment;	property=\ <i>propertyName</i> \	An attachment for a property value; see §3.3.
.../choice;	property=\ <i>propertyName</i> \ - or - action=\ <i>actionName</i> \; param=\ <i>paramName</i> \	A domain object (or scalar value) acting as a choice for a property §C16.4.1 or an action parameter §C18.2.1
.../clear	property=\ <i>propertyName</i> \	Clear a domain object property §C16.3
.../collection		Description of a collection §D24, as linked from a domain type §D22
.../default;	action=\ <i>actionName</i> \; param=\ <i>paramName</i> \	A domain object (or scalar value) acting as a default for an action parameter
.../delete		Link to delete a domain object §C14.3
.../details;	property=\ <i>propertyName</i> \ - or - collection=\ <i>collectionName</i> \ - or - action=\ <i>actionName</i> \	Details of a property §C16.1, collection §C17.1 or action §C18.1, as linked from a domain object §C14.1 or domain service §C15.1.
.../domain-type		Link to a domain type §D22.
.../domain-types		Link to the catalogue of domain types available in the system §D21
.../element		Link to a domain object §C14 from a list returned by an action §B11.
.../element-type		The domain type §D22 which represents the element of a list or collection
.../invoke;	action=\ <i>actionName</i> \ - or - typeaction=\ <i>typeActionName</i> \	Link to invoke a domain object action §C19, or to invoke a domain type action §D27
.../modify	property=\ <i>propertyName</i> \	Link to modify a single domain object property C16.2. (See also the .../update rel).

# Restful Objects

rel	Parameters	Description
.../persist		Link to persist a proto-persistent object §B9.1
.../property		Description of a property §D23, as linked from a domain type §D22
.../remove-from;	collection="collectionName"	Remove from a domain object collection, §C17.4
.../return-type		The domain type §D22 which represents the (return) type of a property, collection, action or param
.../service;	serviceId="serviceId"	A domain service, §C15.1
.../services		The set of available domain services, §B7.1
.../update		Link to modify all properties of a domain object §C14.2.
.../user		The current user, §B6.1
.../value;	property="propertyName" - or - collection="collectionName"	Link to an object §C14 that is the value of a property §C16.1 or held within a collection §C17.1.
.../version		Version of the spec and implementation, §B8.1

## 2.7.2 "type"

The **"type"** json-property indicates the media type §2.4 of the representation obtained if the link is followed. This will always be "application/json" and will (depending on the implementation §B8) have an additional **"profile"** parameter to further describe the representation.

For example:

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

To make examples more readable, throughout the rest of the spec the "urn:org.restfulobjects:repr-types" literal within the profile parameter is abbreviated to "..."; the above example is written as:

```
application/json;profile=".../object"
```



### 2.7.3 "arguments"

---

Sometimes a link represents a resource that requires additional data to be specified. When a representation includes a link to these resources, it may optionally include an **"arguments"** json-property, for example to provide a default value for an action argument.

Note that the client is not obliged to use this information. The representation of arguments is itself well-defined, see §2.9.

### 2.7.4 "value"

---

The optional **"value"** json-property of a link contains the representation that would be returned from following the link.

Currently the spec does not define any functionality that uses this capability. Future versions of this specification may define a syntax to allow clients to request eager loading of links, §E34.4.

## 2.8 "self"

The majority of representations include a "self" link, specifying the resource by which the representation may be obtained again.

For example, the following might be the initial part of a representation of an Order:

```
{
  "links": [
    {
      "rel": "self",
      "href": "http://~/objects/ORD-123",
      "type": "application/json;profile=\".../object\"",
      "method": "GET"
    },
    ...
  ]
}
```

while the following is the initial part of a Customer's firstName property:

```
{
  "links": [
    {
      "rel": "self",
      "href": "http://~/objects/CUS/001/properties/firstName",
      "type": "application/json;profile=\".../object-property\"",
      "method": "GET"
    },
    ...
  ]
}
```

In addition, the invocation of a query-only action (using GET §C19.1) will also have a **"self"** link, this time linking back to the action. This allows clients to copy (bookmark) the action link if they so wish.

There are however two types of representation that do not have a **"self"** link.

The first is a representation of a proto-persistent object or of a view model §2.2, where there is no server-side resource to address.

The second is the representation returned by any action invoked by either a PUT or POST method §C19.2, §C19.3. These have no self link, to minimize the risk of a client repeating the action and inadvertently causing side effects in the system.

## 2.9 Resource argument representation

In many cases the resources defined by the Restful Objects spec require additional data, for example representing either action arguments or object properties.

Restful Objects defines two mechanisms for passing in such arguments. The 'Formal' mechanism may be used in all circumstances. However, for certain specific situations there is the option to use the "Simple" form, which has the advantage of being simpler to construct and easier for a human to read.

---

### 2.9.1 Simple Arguments

---

If a query-only action is being invoked through GET §C19.1, *and* all arguments are scalar values, then the action may be invoked using simple 'param=value' arguments.

For example:

```
GET services/x.TaskRepository/actions/findTasks?tagged=urgent
```

However, if either of these conditions are not true (the action invoked is called using PUT or POST, or if the action takes arguments that are references to other objects) then this simple form cannot be used.

This form of arguments also cannot be used when updating multiple properties §C14.2. For these cases the 'Formal' mechanism must be used §3.1.2.

---

## 2.9.2 Formal Arguments

---

Although simple arguments §2.9.1 are convenient to use, their applicability is limited. For all other cases arguments<sup>17</sup> must be provided using a more formal syntax, either as a single argument node, or as a map or argument nodes:

- resources that require a single value (§C16.2, §C17.2) take a single argument node;
- the action resource methods (§C19.1, §C19.2, §C19.3) take a map of argument nodes;
- the update of multiple properties §C14.2 takes a map of argument nodes (the arguments representing the property values)
- the persist of a new object (§B9) also takes a map-like structure but in this case the map is based on a cut-down version of the object representation, §C14.4)

Treating property values and action arguments in the same way simplifies matters, but it does require that action resources provide a unique name for each of their arguments (rather than merely by a position, as in a list). For implementations that support named parameters this will simply be the parameter name. For implementations that do not support named parameters, the recommendation is to manufacture one either using existing metadata where available (e.g. a UI hint), or otherwise to use the type name of the parameter (string, int etc). If the action takes more than one argument of a given type, then the implementation can disambiguate using integer suffixes (string1, string2 and so on).

Note that the representations defined here, although they may look like the body of HTTP requests, apply to all resources, that is, to GET and DELETE as well as to PUT and POST. Section §2.10 explains the mechanics of how the argument structures defined here are passed to the resource.

### 2.9.2.1 *Argument node structure*

The structure of an argument node fulfils a number of inter-related requirements:

- it allows the value for the argument to be specified;
- if any of the argument values supplied are found to be invalid, it allows the same representation to be returned in the response, with an **"invalidReason"** json-property for those argument(s) that are invalid

If validation is being requested, then the map need only contain arguments for those to be validated; other arguments can be omitted.

---

<sup>17</sup> The term "arguments" is used here in a very general sense, applying both to the providing of values of object properties as well as of action arguments.

Note that the client **can** request validation of a null value by *providing* an argument node, whose value just happens to be null.

Argument nodes take the following structure:

```
{
  "value": ... ,
  "invalidReason": "xxx"
}
```

where:

JSON-Property	Description
<b>value</b>	is the value of the argument (possibly a link)
<b>invalidReason</b>	(optional) is the reason why the value is invalid.

The **"invalidReason"** json-property is intended to be populated by the server, and would be returned by the server as part of its response if one or more the arguments provided was invalid. If the client provides an **"invalidReason"** in its map then this will be ignored by the server.

If the **"value"** is a link to another domain object resource, then only the **"href"** json-property need be specified; for example:

```
{
  "value": {
    "href": "http://~/objects/ABC/123"
  }
}
```

### 2.9.2.2 **Single value arguments (Property, Collection)**

If providing a new value for a property or a collection then a single argument node should be provided.

For example, the following could represent a new value for the "lastName" property of Customer:

```
{
  "value": "Bloggs Smythe"
}
```

If this value was invalid for some reason, then the server would generate a response:

```
{
  "value": "Bloggs Smythe",
  "invalidReason": "Use hyphenated form rather than spaces"
}
```

### 2.9.2.3 **Argument maps (Actions, Properties)**

Action resources (§C19.2, §C19.3) and the PUT Object resource §C14.2 accept arguments only in map form. In the former case the argument nodes are the values of the arguments, in the latter they represent the property values.

## Restful Objects

---

For example, suppose an object has an action `listProducts(Category category, Subcategory subcategory)`. Arguments for actions are provided in map form:

```
{
  "category": {
    "value": {
      "href": "http://~/objects/CGY/BOOK"
    }
  },
  "subcategory": {
    "value": {
      "href": "http://~/objects/SCG/Fiction"
    }
  }
}
```

Similarly, updating multiple properties could be done using the following map:

```
{
  "firstName": {
    "value": "Joe"
  },
  "lastName": {
    "value": "Bloggs"
  },
  "status": {
    "value": {
      "href": "http://~/objects/STS/NEW"
    }
  }
}
```

Only domain object properties that match the json-properties of this map will be updated; json properties that do not match an object property will result in a 400 (syntax error).

### *Providing values for blob/clob properties or arguments*

If a property or argument is a blob or clob (§2.5) then (just like any other datatype) the value can be provided in-line within a map. In the case of a blob, the byte array must be base 64 encoded.

### *Validating individual property/arguments*

If any of the values provided are invalid, then the returned response will indicate this with an **"invalidReason"** json-property.

For example:

```
{
  "firstName": {
    "value": "Joe"
  },
  "lastName": {
    "value": "Bloggs"
  },
  "status": {
    "value": {
      "href": "http://~/objects/STS/NEW"
    },
    "invalidReason":
      "Cannot set customers that have placed orders to 'New' status"
  }
}
```

### 2.9.2.4 *Validating argument sets*

The client can also request the validation of arguments; this is done by providing the reserved **x-ro-validate-only** param (§3.2)<sup>18</sup>.

In the example introduced above, an object has an action `listProducts(Category category, Subcategory subcategory)`. To validate the category by itself (for example, when the user tabs from the category field in the UI), it would provide only the category argument:

```
{
  "category": {
    "value": {
      "href": "http://~/objects/CGY/BOOK"
    }
  },
  "x-ro-validate-only": true
}
```

If the server found that the argument provided was invalid, then it would indicate it in its response using the **"invalidReason"** json-property:

```
{
  "category": {
    "value": {
      "href": "http://~/objects/CGY/BOOK"
    },
    "invalidReason": "not permitted to select from this category "
  }
}
```

### 2.9.2.5 *Obtaining argument choices*

The set of argument choices for a parameter can be found by obtaining a representation of the action resource §C18.1.1.

For example, the list of categories could be returned as:

---

<sup>18</sup> The "x-ro-" prefix is used to distinguish from regular argument names.

```
{
  "category": {
    "...
    "choices": [
      { "href": "http://~/objects/CGY/BOOKS" },
      { "href": "http://~/objects/CGY/ELECTRICAL" },
      { "href": "http://~/objects/CGY/GARDEN" },
      { "href": "http://~/objects/CGY/HOME" },
      { "href": "http://~/objects/CGY/LEISURE" }
    ]
  }
}
```

Note that the spec does not currently support obtaining the set of choices of one parameter based on another; see §E34.5 for discussion on proposals for this as a future feature.

### 2.10 Passing arguments to resources

As noted previously, calling a resource using GET with simple arguments §2.9.1 is straight-forward: the arguments are simply passed as key/value pairs. For example:

```
GET services/x.TaskRepository/actions/findTasks?tagged=urgent
```

Passing formal arguments §2.9.2 through to resources that accept a PUT or a POST is also easy: a string representation of the arguments map should simply be provided as the body of the request.

However, if formal arguments need to be passed through to a resource using GET and DELETE then matters are slightly more complex, because the HTTP spec<sup>19</sup> does not guarantee that resources called using GET and DELETE will receive a body<sup>20</sup>. Therefore, any query arguments to such resources must be encoded within the URL. In the case of a query argument representing a link, this should be converted to its string form first, and then URL encoded. The result is used as the entire query string.

For example, suppose the `OrderRepository#findOrdersPlacedBy` action takes a reference to a customer. The argument representation for this reference:

```
{
  "placedBy": {
    "value": {
      "ref": "http://~/objects/CUS/123",
    }
  }
}
```

---

<sup>19</sup> <http://www.w3.org/Protocols/rfc2616/rfc2616.html>, sec 4.3 and 9.7.

<sup>20</sup> Empirical testing confirms that bodies are not preserved by servlet containers such as Tomcat and Jetty. Proxies may also strip out the body.

## Restful Objects

can be encoded<sup>21</sup> to:

%7B0A%20%20%22placedBy%223A%20%7B%200A%20%20%20%20%22value%223A%20%7B0A%20%20%20%20%20%20%22ref%223A%20%22http%3A%2F%2F~%2Fobjects%2FABC-123%22%2C0A%20%20%20%20%20%7D0A%20%20%7D0A%7D0A

This is appended to the end of the URL, such that the entire URL is:

http://~/services/x.OrderRepository/actions/findOrdersPlacedBy?%7B%0A%20%20%22placedBy%22%3A%20%7B%20%0A%20%20%20%22value%22%3A%20%7B%0A%20%20%20%20%20%20%20%22ref%22%3A%20%22http%3A%2F%2F~%2Fobjects%2FABC-123%22%2C%0A%20%20%20%20%20%20%7D%0A%20%20%7D%0A%7D%0A

## 2.11 Extensible Representations

All of the representations defined by the Restful Objects spec include two json-properties that allow implementations to provide additional (implementation-specific) information in a standardized fashion.

The **"links"** json-property is intended to allow a list of additional links from the representation to other resources. As always for links, the **"rel"** json-property of the link indicates the nature of the resource being linked to. The **"extensions"** json-property, meanwhile, is a map to allow additional data json-properties to be provided.

## 2.12 URL encoding and Case sensitivity

The URLs defined by the Restful Objects spec follow the rules defined by the HTTP spec<sup>22</sup>. In particular, this means that URL matching is case sensitive<sup>23</sup>, and that certain characters (such as "/", "|", "&", ":") may not be used directly, and so must be URL encoded with respect to a particular character set.

Restful Objects requires that all URLs are encoded using UTF-8. All modern implementation languages (Java, .NET, Ruby, Python etc) provide built-in support for URL encoding to this character set.

The character set of JSON representations is not mandated by the spec; instead the response will indicate the character set through the **Content-Type** header; for example:

```
application/json;profile="...";charset=utf-8
```

Unless there is a good reason to do otherwise, it is recommended that implementations use UTF-8.

<sup>21</sup> eg, using <http://meyerweb.com/eric/tools/dencoder/>

<sup>22</sup> <http://www.ietf.org/rfc/rfc1738.txt>;

<sup>23</sup> excluding the hostname part of a URL, which is case insensitive.



### 2.13 Caching (Cache-Control and other headers)

REST-based systems cache representations of certain resources to reduce the number of round-trips. This is analogous to how a web browser might cache images, CSS, or Javascript, without necessarily caching the HTML page itself.

To facilitate this Restful Objects specifies that all responses must indicate whether they may be cached or not. The spec distinguishes three cases:

- **No caching:** suitable for transactional resources such as domain objects and domain object members;
- **Short-term caching:** suitable for user resources that might encapsulate the users' credentials. Such resources might typically be cached for 1 hour (3600 seconds).
- **Long-term caching:** suitable for read-only resources such as domain model resources. Such resources might typically be cached for 1 day or longer (86400 seconds).

Implementations are expected to provide their own configuration settings to allow these values to be tuned. In the remainder of the spec the placeholders "TRANSACTIONAL", "USER\_INFO" and "NON\_EXPIRING" are used:

- "TRANSACTIONAL" is for resources that are frequently updated, for example a Customer;
- "USER\_INFO" is for resources that represent a user's credentials, and so might change over time but not often;
- "NON\_EXPIRING" is for resources that are not expected to change over time

In the spec these placeholders map onto the HTTP 1.1 **Cache-Control** header. In addition, HTTP 1.0 **Pragma**, **Date** and **Expires** headers should also be set in order to support any legacy HTTP 1.0 proxies.

The table below summarizes the values to be set:

Caching	Cache-Control	Pragma	Date	Expires
TRANSACTIONAL (low volume scenario)	non-cache	No-Cache	(current date/time)	0
TRANSACTIONAL (high volume scenario)	max-age: 2		(current date/time)	Date + #seconds
USER_INFO	max-age: 3600		(current date/time)	Date + #seconds

# Restful Objects

Caching	Cache-Control	Pragma	Date	Expires
NON_EXPIRING	max-age: 86400		(current date/time)	Date + #seconds

As can be seen, in a high-volume environment implementations are permitted to specify a small degree of caching for "TRANSACTIONAL" resources in order to support reverse proxying. The means by which the amount of caching is set is implementation-specific.

## 2.14 Security

### 2.14.1 Authentication

Restful Objects currently does not specify any particular approach to user authentication. Instead, it is expected that an out-of-band mechanism (such as *oauth*<sup>24</sup>) is used.

Note, though, that the URLs defined by Restful Objects do not encode the identity of the user requesting the resource. This is deliberate: so that representations may be cached by server-side caching infrastructure<sup>25</sup>.

### 2.14.2 Authorisation ("disabledReason")

Restful Objects defines two mechanisms by which the requesting user's credentials may affect the representations that are returned.

First, if the credentials are such that the object member is hidden/invisible to that user, then that member will be excluded from the representation.

Secondly, if the credentials are such that the object member is visible but disabled, then the representation of the member will exclude any links to resources for mutating that member.

Furthermore, if a member is visible but disabled, then the representation for the disabled member may include an optional "**disabledReason**" json-property to explain why the member is disabled. The client may choose to render this information in its user interface (for example as a 'tooltip').

---

<sup>24</sup> <http://oauth.net>

<sup>25</sup> assuming, that is, that **Cache-Control** header is not set to no-cache.

Because the URLs defined by Restful Objects are well-defined, there is nothing to prevent a rogue client from guessing URLs and attempting to call them. If the client attempts to access a hidden object member directly (using any HTTP method), then a **404** "not found" will be returned. Or, if the user attempts to mutate a disabled object member using PUT, DELETE or POST, then a **403** "forbidden" will be returned.

### 2.15 Concurrency Control (If-Match, ETag)

Restful Objects defines concurrency control through a combination of the **ETag** HTTP response header and the **If-Match** request header.

The **ETag** header provides a unique digest (typically based on a timestamp for the last time that an object was modified). When a client wishes to perform a (PUT, DELETE or POST) request that will modify the state of a resource, it must also provide the **If-Match** header to indicate the timestamp of the representation that it previously obtained from the server.

If the object has been modified since that time, then a **412** "Precondition failed" status code will be returned.

If the client fails to provide the **If-Match** header, then the response will be **400** "Bad Request", with an appropriate **Warning** header.

If the domain object does not have timestamp information (for example, if it is immutable), then no **ETag** header need be (nor sensibly can be) generated. For these resources, the **If-Match** header should not be provided by the client (but if it is, then the server will simply ignore it rather than return an error return code).

Restful Objects does not require that the **If-Modified** response header is provided in representations (though implementations are free to return it if they wish). Note that **If-Modified** is not appropriate for concurrency control because its precision is only to the nearest second.

### 2.16 Business Logic Warning and Error

When an action is invoked the business logic may raise an informational, warning or error message. The client may in turn display a warning dialog in the UI.

To support this, Restful Objects allows that the standard "**Warning**" HTTP header can be set. The HTTP status code indicates whether this message should be considered as information (**200**), or a warning (**4xx** or **5xx**).

### 2.17 Malformed JSON Representations

The correct form for JSON representations is:

## Restful Objects

---

```
{  
  "foo": "bar",  
  "baz": "boz"  
}
```

However, some REST APIs and implementations incorrectly serve malformed JSON, where the keys are not quoted:

```
{  
  foo: "bar",  
  baz: "boz"  
}
```

Implementations of Restful Objects must always serve up correctly formed JSON representation. However, where a client posts JSON to the server (for example, to modify a resource), the implementation must accept malformed JSON representations where the key has not been quoted<sup>26</sup>.

---

<sup>26</sup> This is an application of Postels' law: be conservative in what you do, be liberal in what you accept from others; <http://tools.ietf.org/html/rfc793>.

### 3 OPTIONAL CAPABILITIES

While Restful Objects aims to define a consistent standard for RESTful interactions with domain models, some useful features might be easy for one framework to accomplish, but much more difficult for another. Therefore the specification defines a small number of capabilities that are optional.

Implementations advertise the capabilities that they support through the "version" resource, §B8.

In some cases, clients can vary the behaviour of these capabilities by providing optional query parameters. For example, if argument validation is supported by the implementation, then the client can use the "**x-ro-validate-only**" query parameter to suppress modification of the resource, and simply check values.

In order to minimize clashes with other (application) parameters, the optional query parameters all have the "**x-ro-**" prefix: the "x-" is intended to indicate a non-standard parameter; the "ro-" to indicate that the parameter is specific to Restful Objects.

If a framework has not implemented some optional aspect of the Restful Objects specification and can reasonably continue, then it should ignore the request. If there is no reasonable way to continue, then the framework should return a **501** "Not implemented" status code along with a **Warning** header explaining the feature that has not been implemented.

The sections that follow each indicate the query parameter that is used to request the capability.

#### 3.1 Domain Metadata (x-ro-domain-model)

Some clients may wish to perform client-side validation before submitting changes to the server: examples include the enforcement of mandatory properties (or action parameters) and the enforcement of maximum string length. Such rules are applicable to any domain object instance of that given type, and so may be defined on the domain type

Restful Objects defines two ways in which such domain type information may be represented: a "simple" scheme and a "formal" scheme, defined below. Common to both is that the information is accessible by way of links and extensions §2.11.

A client may query the version resource §B8 to determine the server's support for domain metadata:

- a value of "none" indicates that the implementation does not provide domain type information;
- a value of "simple" or "formal" means that the server supports only that scheme

## Restful Objects

- a value of "selectable" is for implementations that support both schemes. By default such implementations will return the "simple" scheme, but the client can provide a reserved **x-ro-domain-model** query parameter to request "formal".

### 3.1.1 Simple Scheme

In the simple scheme, Restful Objects allows that implementations may inline certain domain type information within the "**extensions**" json-property of the domain object representation.

For example, the fact that a property is required (may not be left empty) is captured using:

```
{
  ...
  "extensions": {
    "optional": false,
    ...
  }
}
```

Restful Objects defines the following standard json-properties for the "simple" scheme:

JSON-Property	Values	Applies to	Description
<b>domainType</b>	string	domain object	unique identifier for the domain type
<b>friendlyName</b>	string	domain object, property, collection, action, action param	Version of the name suitable for use in a UI (e.g. as a label).
<b>pluralName</b>	string	domain object	Pluralized form of the friendly name, for use in a UI (e.g. as a label of a collection).
		collection action returning list	Pluralized form of the element type within the collection/list.
<b>description</b>	string	domain object, property, collection, action, action param	Description, suitable for use in a UI (e.g. as a tooltip).
<b>isService</b>	boolean	domain object	whether this domain object is a domain service

## Restful Objects

JSON-Property	Values	Applies to	Description
<b>optional</b>	boolean	property, action param.	if false, then a value for the property / param must be provided. Default is implementation-specific.
<b>maxLength</b>	int	string property, string action param	the maximum number of characters that the string may contain. A value of 0 means unlimited.
<b>pattern</b>	string	string property, string action param	whether value must match the regular expression that any submitted value must match.
<b>returnType</b>	string	property action (returning scalar or object) action param	If scalar value returned, indicates its datatype §2.5 (in conjunction with 'format' if 'string'). If object returned, its domain type id.
		collection action (returning collection)	Either 'list' or 'set'.
		action returning void	'void'
<b>format</b>	string	when returnType if 'string'	further qualifies the datatype §2.5.
<b>elementType</b>	string	collection action returning collection	of the domain type id for the type of the elements held within the collection.
<b>hasParams</b>	boolean	action	whether an action has parameters or not. This may, for example, be used by clients to render ellipsis (...) in their UI.
<b>memberOrder</b>	int	property, collection, action	a presentation hint indicating the recommended relative display order for each member. Discussed further below.

Implementations are free to extend this list as they require.

### *MemberOrder*

The "**memberOrder**" json-property is a presentation hint indicating the recommended display order for each member of the object relative to others. Note that clients are not obliged to adhere to member ordering.

Irrespective of whether it is used, the "**memberOrder**" json-property *must* always be provided by the implementation. However, in the cases where no ordering information is available, the implementations may provide the same value for more than one member. For example, an implementation might return 0 for all unordered members (putting them joint first place in the list) or it might return a value of 999, say (putting them all joint last place).

Alternatively, an implementation may choose to synthesise ordering information, for example based on the declaration order of its source code, or on the alphabetic order of the member names.

---

### 3.1.2 Formal Scheme

---

The formal scheme of providing domain type information defines separate resources that generate representations of the domain metamodel. If this scheme is followed then such resources are obtained by **rel="describedby"** link in the "**links**" json-property.

For example, suppose that the Customer class has an (int) "id" property, a date "since" property, and a "blacklist" action.

In .NET, this could be written as:

```
public class Customer {  
    ...  
    public int Id {get; set; }  
    public DateTime Since {get; set; }  
    public bool Blacklist(string reason) { ... }  
    ...  
}
```

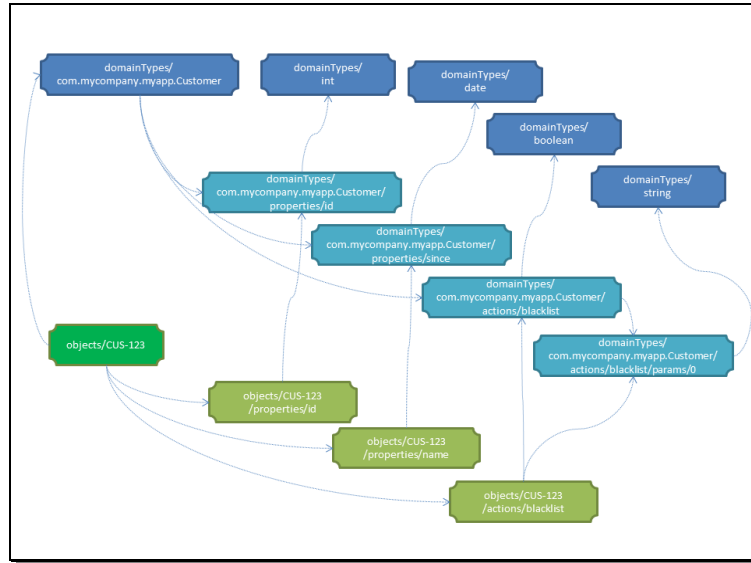
while in Java it might look like:

```
public class Customer {  
    ...  
    private int id;  
    private Date since;  
  
    public int getId() { return this.id; }  
    public void setId(int id) { this.id = id; }  
  
    public Date getSince() { return this.since; }  
    public void setSince(Date since) { this.since = since; }  
  
    public boolean blacklist(String reason) { ... }  
    ...  
}
```



# Restful Objects

The resources to expose the metadata for an instance of this class are shown in §D.



**FIGURE 3: DOMAIN OBJECTS VS DOMAIN TYPES**

Domain type resources are pre-defined for the scalar types §D21.3. For non-scalar types, the domain type is the concatenation of "http://~/domain-types/" + the domain type id.

The link to the domain type resource is shown in the domain object representation as:

```
{
  "links": [ {
    "rel": "describedby",
    "href": "http://~/domain-types/x.Customer",
    "type": "application/json;profile=\".../domain-type\"",
    "method": "GET"
  },
  ...
]
...
}
```

where the referenced domain type resource will return a representation that describes the domain object instance.

## *Restricting access to formal metadata*

Implementations that implement the formal scheme should be aware that there is a potential security risk: clients will be able to ascertain the existence of an object's members, even if the member is not returned in any representation that they obtain of that member.

For example, an Employee object might have a salary property that is only visible to users with certain permissions (e.g. an "HR" role). An ordinary user browsing representations of Employee objects would be able to view the name and phone, but the salary would be hidden from view. However, navigating to the formal domain type resource would show that a salary property does exist.

Because domain types are intended to be cacheable, implementations should not attempt to alter the metadata representations on a user-by-user basis. If an implementation intends to support use cases where the above issue might be considered a security risk, then it should also offer the simple scheme and provide an implementation-specific mechanism to disable formal scheme support.

### 3.2 Validation (x-ro-validate-only)

If validation logic has been defined for a property value, a collection reference, or an action's parameter(s), then the server implementation is expected to perform that validation prior to initiating any change. For example, a Customer's firstName property might disallow certain characters, or its showPayments() action might require that the toDate parameter is greater than the fromDate.

A validation failure will generate a **422** "unprocessable entity" status code, and in addition, a warning message will be returned. This will either be a simple **Warning** header, or, dependent on the request, may be part of the response, in the form of an **"invalidReason"** json-property.

*x-ro-validate-only reserved query parameter*

On occasion a client may want to validate one or more property fields, before attempting to modify an object, or may want to validate arguments before attempting to invoke the action.

Restful Objects defines an optional capability §B8 whereby the client can set the reserved **x-ro-validate-only** query param for the request to indicate that only validation should be performed:

If the validation completes, then a **204** "No content" status code will be returned. If a validation failure occurs, then the response will be **422** "unprocessable entity" with corresponding **Warning** header / **"invalidReason"** json-properties.

### 3.3 Blobs/Clobs and Attachments

As well as properties representing strings and dates, etc, the specification also defines optional support for properties whose value is a blob (binary large object) or a clob (character large object) §2.5. A typical example is a property representing a media item such as a picture or document.

If an implementation does support blobs/clobs, then the value of the blob/clob is suppressed from the property representation. Instead, the representation includes a **"rel"**=".../attachment;" link. If followed, such a link returns a representation with the appropriate content-type, e.g. image/jpeg, application/pdf, etc

For example, if a property is a blob representing an image, then its representation would include a link with a corresponding attachment:

```
{
  "links": [
    {
      "rel": ".../attachment;property=\"photo\"",
      "href": "http://~/objects/CUS/123/properties/photo",
      "type": "image/jpeg",
      "method": "GET"
    }
    ...
  ]
}
```

The href of this link should be the same as the property resource §C16.1, however the client should provide a different **Accept** header in order to obtain the attachment.

The values of blob or clob properties are set/cleared using PUT (§C16.2) and DELETE (§C16.3), as for any other property. If attachments are supported then the PUT resource for the property representation C16.2 may also accept the value in its native media type (e.g. image/jpeg).

A client can determine whether an implementation supports blobs/clobs by inspecting the version resource §B8.

### 3.4 Proto-persistent Objects

As described in §2.2, a proto-persistent domain entity is an object instance that is created as a result of an interaction and immediately represented back to the client, without having been persisted first.

The ultimate persistence of the entity is therefore under the control of the client, which is done by POSTing to the Objects of Type resource, §B9.1.

Support for proto-persistent objects is an optional capability because providing a general-purpose persistence capability may not be practicable for some implementations.

### 3.5 Object Deletion

Persisted objects can be deleted through the DELETE Object resource, §C14.3.

## Restful Objects

---

This is an optional capability because implementing a generic 'delete object' capability - which includes managing any references to the deleted object throughout the system - is potentially complex, and not necessarily practicable for many implementations.

If the implementation does support the capability then it must also determine that it is safe to delete the object. A **405** ("method not allowed") error will be returned otherwise.

## 4 SPECIFIED ELEMENTS

This section summarises the standard json-properties, headers and status codes that are understood by Restful Objects implementations.

### 4.1 Specified json-properties

There are a number of json-properties specified by Restful Objects that have well-defined and fixed meanings, irrespective of which representation they appear within. They are:

JSON-Property	Description
<b>disabledReason</b>	Provides the reason (or the literal "disabled") why an object property or collection is un-modifiable, or, in the case of an action, unusable (and hence no links to mutate that member's state, or invoke the action, are provided).
<b>invalidReason</b>	Provides the reason (or the literal "invalid") why a proposed value for a property, collection or action argument is invalid. Appears within an argument representation §2.9 returned as a response.
<b>x-ro-invalidReason</b>	Provides the reason why a <b>SET OF</b> proposed values for properties or arguments is invalid. The "x-ro-" prefix is to avoid name clashes with the property/argument names.
<b>links</b>	A list of additional links from the representation to other resources. Implementation-specific links may also be present in this list; see §2.11.
<b>extensions</b>	Map of additional information about the resource. Implementation-specific json-properties may also be present in this map; see §2.11.

### 4.2 Specified (reserved) query parameters

The query parameters reserved by Restful Objects are:

Header	Description
<b>x-ro-domain-model</b>	Which domain model scheme §3.1 the server should return. Understood only by servers that provide "selectable" for the "domainModel" capability §B8.
<b>x-ro-validate-only</b>	Indicates that parameters should be validated but no change in state should occur.

### 4.3 Specified headers

Restful Objects defines both request and response headers.

## 4.3.1 Request headers

The request headers specified by Restful Objects are:

Header	Description
Accept	The list of media types accepted by the client.
If-Match	The value of the <b>ETag</b> response header for the most recently obtained representation of a resource.

## 4.3.2 Response headers

The response headers specified by Restful Objects are:

Header	Description
Allow	The HTTP methods that are supported by the resource. Returned only in conjunction with a 406 ("Not allowed")
Cache-Control	Whether the representation may be cached or not. Representations of resources representing domain objects will typically disable caching, but some representations (for example, of immutable objects, or of domain types) may be cached.
Last-Modified	The last modified timestamp of a persistent resource. The value of this should be passed as the If-Unmodified-Since header.
Warning	Header to describe either any errors returned by the server implementation, or, as raised by the domain object business logic (e.g. if the request was syntactically valid but could not be completed).
Content-Type	Depends upon representation; in the form "application/json;profile=http://restfulobjects.org/xxx".

## 4.4 Specified status return codes

The status return codes specified by Restful Objects are:

Code	Description	When
200	Success	Successfully generated representation
204	No content	Request was successful but generated no representation; or validation (x-ro-validate-only) succeeded. Note however that invoking a void action DOES return a representation §C19.4.4.
400	Bad request	Represents any of: a syntactically invalid request, missing mandatory information (e.g. If-Unmodified-Since header), or a warning message generated by the application.

## Restful Objects

Code	Description	When
401	Not authorized	Problem with user's credentials (ie user is not authenticated).
403	Forbidden	User is not authorized to invoke resource (modify property or collection, or invoke action)
404	Not found	Property, collection or action not found (could be that the member is hidden for the requesting user)
405	Method not allowed	An attempt was made to invoke a resource with an HTTP method that is not supported by that resource.
406	Not acceptable	Content type of the representation that would be returned is incompatible with the provided <b>Accept</b> header (in other words, the server is unable to comply with the accept instruction).
412	Precondition failed	Concurrency error; the object has been modified and its current etag does not match that provided in the If-Match header.
500	Internal server error	Indicates that the domain object threw an exception in its business logic
501	Not implemented	This implementation of Restful Objects does not support the feature requested.

All client- and server-side errors (4xx and 5xx) will also result in a **Warning** header being returned to describe the nature of the problem. The format of this Warning should be<sup>27</sup>:

199 RestfulObjects xxx

where:

- 199 indicates a miscellaneous "warn-code" (as per HTTP/1.1 spec)
- RestfulObjects is the "warn-agent"
- xxx is the text of the message generated by the implementation

In some cases the implementation will be able to provide a detailed error message; otherwise it should return a standard generic message. The sections describing resource responses detail these messages.

---

<sup>27</sup> <http://www.ietf.org/rfc/rfc2616.txt>





# B

## SUPPORTING RESOURCES AND REPRESENTATIONS



## 5 HOME PAGE RESOURCE & REPRESENTATION

The 'home page' is a well-known resource which acts as the starting point for any client. From it all other resources may be discovered.

### 5.1 HTTP GET

Obtain the representation of the home page resource.

The endpoint URL for this resource is:

/

(in other words the base directory).

---

#### 5.1.1 Request

##### 5.1.1.1 *Query String*<sup>28</sup>

None

##### 5.1.1.2 *Headers*

- **Accept**
  - application/json
  - application/json;profile=.../homepage

##### 5.1.1.3 *Body*

- N/A

---

#### 5.1.2 Successful Response

##### 5.1.2.1 *Status Code*

- 200 "OK"

##### 5.1.2.2 *Headers*

- **Content-Type**
  - application/json;profile=".../homepage"
- Caching headers:
  - NON\_EXPIRING, see §A2.13
    - since home page changes only on redeployment

---

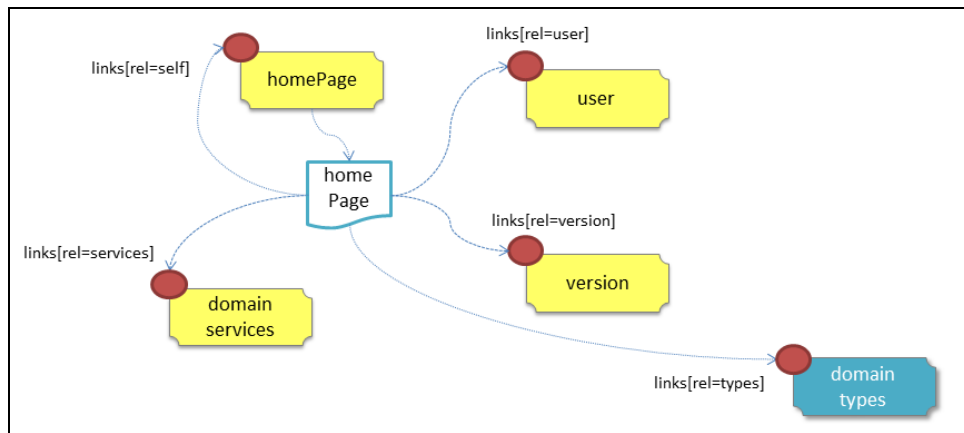
<sup>28</sup> [http://en.wikipedia.org/wiki/Query\\_string](http://en.wikipedia.org/wiki/Query_string)

## 5.1.2.3 Body

As per §5.2.

## 5.2 Representation

The links from the home page representation to other resources are as shown in the diagram below:



**FIGURE 4: HOME PAGE REPRESENTATION**

The link to the domain types resource is only present if the formal scheme (§A3.1.2) capability is supported.

The JSON representation is as follows:

```
{
  "links": [ {
    "rel": "self",
    "href": "http://~/",
    "type": "application/json;profile=\"../homepage\"",
    "method": "GET"
  }, {
    "rel": ".../user",
    "href": "http://~/user",
    "type": "application/json;profile=\"../user\"",
    "method": "GET"
  }, {
    "rel": ".../services",
    "href": "http://~/services",
    "type": "application/json;profile=\"../list\"",
    "method": "GET"
  }, {
    "rel": ".../types",
    "href": "http://~/types",
    "type": "application/json;profile=\"../types\"",
    "method": "GET"
  } ]
}
```

## Restful Objects

---

```
{
  "rel": ".../version",
  "href": "http://~/version",
  "type": "application/json;profile=\".../version\"",
  "method": "GET"
}, {
  "rel": ".../domain-types",
  "href": "http://~/domain-types",
  "type": "application/json;profile=\".../type-list\"",
  "method": "GET"
},
...
],
"extensions": { ... }
}
```

where:

JSON-Property	Description
<b>links</b>	list of links to resources.
<b>link[rel=self]</b>	link back to the resource that generated this representation.
<b>link[rel=.../user]</b>	link to the user resource §6
<b>link[rel=.../services]</b>	link to the services resource §7
<b>link[rel=.../version]</b>	link to the version resource §8
<b>link[rel=.../types]</b>	link to the domain types resource §D21
<b>extensions</b>	additional information about the resource.

Restful Objects defines no standard json-properties for "**extensions**". Implementations are free to add to their own links/json-properties to "**links**" and "**extensions**" as they require.



# 6 USER RESOURCE & REPRESENTATION

The 'user' resource represents the currently logged-in user.

The endpoint URL for this resource is:

`/user`

## 6.1 HTTP GET

Obtain representation of the currently logged-in user.

---

### 6.1.1 GET Request

#### 6.1.1.1 Query String

None

#### 6.1.1.2 Headers

- **Accept**
  - `application/json`
  - `application/json;profile=".../user "`

#### 6.1.1.3 Body

- N/A

---

### 6.1.2 GET Successful Response

#### 6.1.2.1 Status Code

- 200 "OK"

#### 6.1.2.2 Headers

- **Content-Type**
  - `application/json;profile=".../user"`
- Caching headers: SHORT, see §A2.13
  - temporary caching of user details

#### 6.1.2.3 Body

As per §6.2.

## 6.2 Representation

The links from the user representation to other resources are as shown in the diagram below:

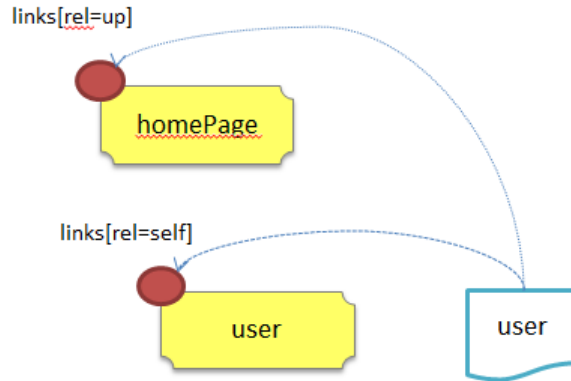


FIGURE 5: USER REPRESENTATION

The JSON representation is:

```
{
  "links": [ {
    "rel": "self",
    "href": "http://~/user",
    "type": "application/json;profile=\".../user\"",
    "method": "GET"
  }, {
    "rel": "up",
    "href": "http://~/",
    "type": "application/json;profile=\".../homepage\"",
    "method": "GET"
  },
  ...
],
  "userName": "joebloggs",
  "friendlyName": "Joe Bloggs",
  "email": "joe@bloggs.com",
  "roles": [
    "role1", "role2", ...
  ],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
links	list of links to resources.
links[rel=self]	link to a resource that can generate this representation
links[rel=up]	link to the homepage resource, §5.
userName	a unique user name
friendlyName	(optional) the user's name in a form suitable to be rendered in a UI.



## Restful Objects

---

JSON-Property	Description
<b>email</b>	(optional) the user's email address, if known
<b>roles</b>	list of unique role names that apply to this user (may be empty).
<b>extensions</b>	additional metadata about the resource.

Restful Objects defines no standard json-properties for "**extensions**". Implementations are free to add to their own links/json-properties to "**links**" and "**extensions**" as they require.



# 7 DOMAIN SERVICES RESOURCE

Returns a list §11 of (links to) domain service resources §C15.

The endpoint URL for this resource is:

/services

## 7.1 HTTP GET

---

### 7.1.1 GET Request

---

#### 7.1.1.1 Query String

None

#### 7.1.1.2 Headers

- **Accept**
  - application/json
  - application/json;profile=".../list "

#### 7.1.1.3 Body

- N/A

### 7.1.2 GET Successful Response

---

#### 7.1.2.1 Status Code

- 200 "OK"

#### 7.1.2.2 Headers

- **Content-Type**
  - application/json;profile=".../list ";x-ro-element-type="yyy"
    - where "yyy" is the domain type id for the services (most likely a type representing a common base class for all services, eg Object)
- Caching headers:
  - NON\_EXPIRING, see §A2.13
    - list of available services will not change between deployments

#### 7.1.2.3 Body

As per §7.2.

## 7.2 Representation

The returned representation is a simple list §11, but with an additional link with a rel="up" referring back to the homepage resource §5.

For example:

```
{
  "links" : [ {
    "rel": "self",
    "href" : "http://~/services",
    "method" : "GET",
    "type" : "application/json;profile=\".../services\""
  }, {
    "rel": "up",
    "href": "http://~/",
    "type": "application/json;profile=\".../homepage\"",
    "method": "GET"
  },
  ...
] ],
  "value" : [ {
    "rel": ".../service;serviceId=\"toDoItems\"",
    "href" : "http://~/services/toDoItems",
    "method" : "GET",
    "type" : "application/json;profile=\".../object\"",
    "title": "To Do Items"
  }, {
    "rel": ".../service;serviceId=\"categories\"",
    "href" : "http://~/services/categories",
    "method" : "GET",
    "type" : "application/json;profile=\".../object\"",
    "title": "Categories"
  },
  ...
] ],
  "extensions": { ... }
}
```

The links from the services representation to other resources are as shown in the diagram below:

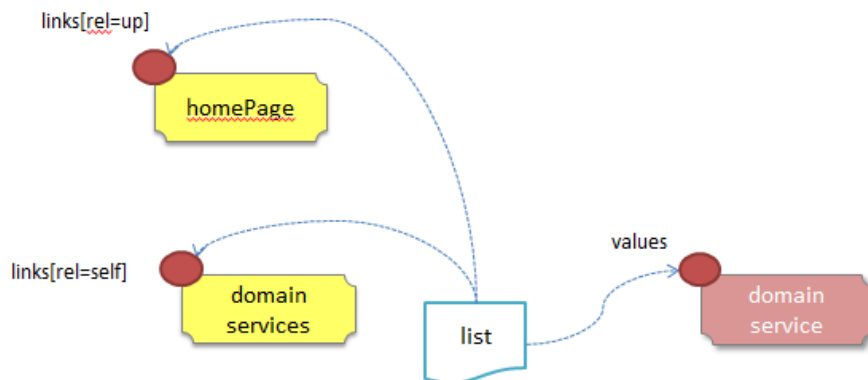


FIGURE 6: SERVICES REPRESENTATION

## Restful Objects

---

Note that the links in this list will point to domain service resources §C15, rather than to domain object resources §C14.



## 8 VERSION RESOURCE & REPRESENTATION

This resource allows a client to dynamically determine which version of the Restful Objects spec a particular implementation supports, the version of the implementation (code) itself, and which of the optional capabilities §A3 it provides.

The endpoint URL for this resource is:

`/version`

### 8.1 HTTP GET

Obtain a representation of the implementation's version and optional capabilities.

---

#### 8.1.1 GET Request

##### 8.1.1.1 Query String

- none

##### 8.1.1.2 Headers

- **Accept**
  - application/json
  - application/json;profile=".../version"

##### 8.1.1.3 Body

- N/A

---

#### 8.1.2 GET Response

##### 8.1.2.1 Status Code

- 200 "OK"

##### 8.1.2.2 Headers

- **Content-Type**
  - application/json;profile=".../version"
- Caching headers:
  - NON\_EXPIRING, see §A2.13
    - version and capabilities will not change between deployments

##### 8.1.2.3 Body

As per §8.2.

## 8.2 Representation

The links from the version representation to other resources are as shown in the diagram below:

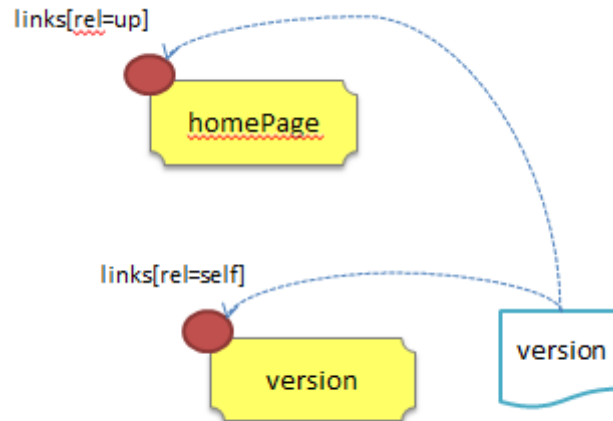


FIGURE 7: VERSION REPRESENTATION

The JSON representation is:

```
{
  "links": [ {
    "rel": "self",
    "href": "http://~/version",
    "type": "application/json;profile=\".../version\"",
    "method": "GET"
  }, {
    "rel": "up",
    "href": "http://~/",
    "type": "application/json;profile=\".../homepage\"",
    "method": "GET"
  },
  ...
],
  "specVersion": "0.69.0",
  "implVersion": "xxx",
  "optionalCapabilities": {
    "blobsClobs": "attachments",
    "deleteObjects": "no",
    "domainModel": "formal",
    "protoPersistentObjects": "yes",
    "validateOnly": "no"
  },
  "extensions": {
    ...
  }
}
```

where:

JSON-Property	Description
links	list of links to resources.



# Restful Objects

JSON-Property	Description
<b>links[rel=self]</b>	link to a resource that can generate this representation.
<b>links[rel=up]</b>	link to the home page resource, §5.
<b>specVersion</b>	Version of the Restful Objects spec supported by this implementation e.g. "1.0.0"
<b>implVersion</b>	(optional) Version of the implementation itself (format is specific to the implementation).
<b>optionalCapabilities</b>	map representing the optional capabilities §A3 supported by this implementation (see below)
<b>extensions</b>	additional metadata about the resource.

## *"optionalCapabilities"*

The "**optionalCapabilities**" json-property holds a map of child properties describing the functionality supported by the implementation.

Capability	Value type	String value	The implementation's support for...
<b>blobsClobs</b>	boolean	---	blobs/clobs see §A3.3.
<b>deleteObjects</b>	boolean	---	deletion of persisted objects through the DELETE Object resource §C14.3, see §A3.5
<b>domainModel</b>	string	none simple formal selectable	different domain metadata representations. A value of "selectable" means that the reserved <b>x-domain-model</b> query parameter is supported, see §A3.1
<b>protoPersistentObjects</b>	boolean	---	proto-persistent objects are supported, see §A3.4
<b>validateOnly</b>	boolean	---	the reserved <b>x-ro-validate-only</b> query parameter, see §A3.2

## *"links" and "extensions"*

Restful Objects defines no standard links/json-properties for "**links**" and "**extensions**", but implementations are free to add to their own links/json-properties as they require.



# 9 OBJECTS OF TYPE RESOURCE

The "Object of Type " resource is the target of a persist link to persist a proto-persistent ("not yet persisted") object §A2.2 of a given domain type.

The overall process is:

- client invokes an action that creates a proto-persistent object representation
- client uses the "**arguments**" json-property of the "**rel**"= ".../persist" link to determine the information required
- client obtains the required information, for example, prompting for it in a user interface
- client posts the arguments map back to the Objects resource, with the missing information
- assuming that the values are valid, a representation of the newly persisted domain object is returned
  - the response code is 201, and will have the **Location** header
  - the representation itself will now include the "**self**" link, the "**instanceId**" json-property (and the "domainType" json-property if simple scheme §A3.1.1).

The endpoint for this resource is:

/objects/{domainType}

where:

- {domainType} uniquely identifies the domain type of the objects being persisted

## 9.1 HTTP POST

Persist a domain object by posting a cut-down version of its representation.

---

### 9.1.1 POST Request

---

#### 9.1.1.1 Headers

- **Accept**
  - application/json
  - application/json;profile=".../object"

## 9.1.1.2 **Body**

Because this resource is in a sense "uploading" a new object, the body is the cut-down version of the domain object representation §C14.4.<sup>29</sup> It consists of:

- members[memberType=property]

In other words, it includes all properties and their values (including those that would normally be hidden), along with a reference to the domain type of the object being persisted.

In addition, it may include the reserved query parameter:

- **x-ro-validate-Only**
  - "true"
    - validate that the representation provided could be persisted as a new object, without actually persisting it.

For example:

```
{
  "members": {
    "firstName": {
      "value": ...
    },
    "lastName": {
      "value": ...
    },
    ...
  }
}
```

---

## 9.1.2 **POST Successful Response**

---

As per §C13.2 (201), returning a domain object representation §C14.4.

---

## 9.1.3 **POST Validation Only and Succeeded**

---

### 9.1.3.1 **Status Code**

- 204 "No content"

### 9.1.3.2 **Headers**

- none

### 9.1.3.3 **Body**

- none

---

<sup>29</sup> Note that this is different from the body provided to PUT Object §C14.2 used to update multiple properties.

---

## 9.1.4 POST Invalid Request

---

For example, if the body is malformed, or any other syntax error.

### 9.1.4.1 Status Code

- 400 "bad request"

### 9.1.4.2 Headers

- **Warning**
  - summary message – indicating the nature of the error

### 9.1.4.3 Body

Error representation, § 10.

---

## 9.1.5 POST Validation Failed Response

---

### 9.1.5.1 Status Code

- 422 "unprocessable entity"

### 9.1.5.2 Headers

- **Warning**
  - summary message

### 9.1.5.3 Body

Body is the same as that posted, but indicating the properties that were invalid, with a reason in each case.

For example:

```
{
  "members": {
    "firstName": {
      "value": "Joe",
    }, {
      "lastName": {
        "value": null
        "invalidReason": "Mandatory"
      },
      ...
    ]
  }
}
```



## 10 ERROR REPRESENTATION

The Error representation defines a standard means for returning detailed diagnostics, typically when a server-side error (status code 500) occurs.

For example, the following might be the result of invoking an action where an exception occurred:

```
{
  "message": "IllegalStateException",
  "stackTrace": [
    "at SomeFile.java#foo():1234",
    "at SomeOtherFile.java#bar():4321"
  ],
  "causedBy": {
    "message": "IllegalStateException",
    "stackTrace": [
      "at LibraryFile.java#foz():567",
      "at LibraryOtherFile.java#baz():765"
    ]
  },
  "links": [ ... ],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
message	the exception message, typically as generated by the underlying implementation programming language.
stacktrace	(optional) list of strings representing call stack
causedBy	(optional) underlying cause of the exception (to handle nested exception scenarios).
links	list of links to other resources
extensions	additional information about the resource

### *"stacktrace"*

The stack trace is optional. Implementations are free to suppress this information if they wish, or to make available only if a debug flag (or similar) has been enabled.

### *"links" and "extensions"*

Restful Objects defines no standard links/json-properties for **"links"** and **"extensions"**, but implementations are free to add to their own links/json-properties as they require.





# 11 LIST REPRESENTATION

The List representation consists of a list of links to domain object §C14 or domain service §C15 resources. Typically it is obtained as the result of invoking a query action, when it is in-lined within the action result representation §C19.4. It is also be obtained directly from the Services resource §B7, when it provides a list of links to domain service resources; . the only difference is that the "**href**" will be a link to a domain service resource §C15 rather than a domain object resource §C14.

The media type for a list representation will always be one of:

- application/json;profile=".../list"
  - for a list of references to domain services
- application/json;profile=".../action-result";x-ro-element-type="..."
  - when in-lined within an action result (ie the representation of an action invocation that returned a list)

Lists of references to domain objects also appear in the object-collection representation; that representation is described in full in §17.5.

The representation itself consists of the following json-properties:

```
{
  "value": [
    ...
  ],
  "links": [ {
    "rel": "self",
    ...
  }, {
    "rel": ".../element-type",
    ...
  },
  ],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
<b>links</b>	list of links to other resources.
<b>links[rel=self]</b>	(optional) link to the resource that generated this list. Present only for domain services.
<b>links[rel=.../element-type]</b>	link to the domain type for the elements within the list – if the “domainModel” optional capability §B8 is implemented.
<b>value</b>	the actual list of links to the domain object resources (or domain service resources)
<b>extensions</b>	additional information about the resource.

A "**self**" link will be present only if the representation was generated from the domain services resource §7.

## Restful Objects

---

Restful Objects defines no standard child properties of the “**extensions**” json-property. Implementations are free to add to their own links/json-properties to “**links**” and “**extensions**” as they require.

# 12 SCALAR VALUE REPRESENTATION

The scalar value representation represents a single value that is not itself a domain object.

This representation is never returned directly from a resource, but may appear in-lined within the representation of an action invocation §C19.4 if the action returned a scalar type such as a string or an integer.

The media type for a scalar representation will always be one of:

- application/json;profile=".../action-result"
  - for the representation of an action invocation that returned a scalar value

The representation itself consists of the following json-properties:

```
{
  "value": ...,
  "links": [ {
    "rel": ".../return-type",
    ...
  },
  ...
],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
<b>value</b>	the scalar value itself.
<b>links</b>	list of links to other resources.
<b>links[rel=.../return-type]</b>	(optional – only if the "formal" scheme §A3.1.2 capability is supported) link to the domain type §D21.3 of the value
<b>extensions</b>	additional metadata about the representation.
<b>extensions.format</b>	(optional – and only if the "simple" scheme §A3.1.1 capability is supported) qualifies the datatype value §A2.5.

Note that there is no "**self**" link, because the scalar value is not an addressable resource.

Restful Objects defines no standard child properties for the "**extensions**" json-property. Implementations are free to add to their own links/json-properties to "**links**" and "**extensions**" as they require.



# C

## DOMAIN OBJECT RESOURCES & REPRESENTATIONS



# 13 RESPONSE SCENARIOS

When a domain object resource is invoked, one of several responses can occur. These can be distinguished by the HTTP status return code

- 200 – Request succeeded, and generated a representation
- 201 – Request succeeded, and generated a representation of a new object
- 204 – Request succeeded, but generated no representation; or validation succeeded
- 400 – Syntactically invalid request
- 403 – Forbidden (user not authorised to modify the resource)
- 404 – Object or object member not found or not visible
- 405 – Method not valid for the resource
- 406 – Client accepts only media types not generated by resource.
- 412 – Pre-condition failed (object changed by another user)
- 422 – Validation failed
- 428 – Required precondition header missing
- 500 – Domain logic failed

In addition, a 401 code may be returned for any resource if the user's credentials are invalid (that is, they have not authenticated themselves).

The following indicates which codes may be returned by a resource:

	Method	Res.	Repr.	200	201	204	400	403	404	405	406	412	422	428	500
Service	GET	15	B11	y			y		y		y				y
Object	GET	C14.1	14.4	y			y		y		y				y
	PUT	C14.2	---	y			y	y	y		y	y	y	y	y
	DELETE	C14.3	---			y	y	y	y	y	y	y	y	y	y
Prop'y	GET	C16.1	16.4	y			y		y		y				y
	PUT	C16.2	---	y			y	y	y		y	y	y	y	y
	DELETE	C16.3	---	y			y	y	y		y	y	y	y	y
Coll'n	GET	C17.1	17.5	y			y		y		y				y
	PUT	C17.2	---	y			y	y	y	y	y	y	y	y	y
	POST	C17.3	---	y			y	y	y	y	y	y	y	y	y
	DELETE	C17.4	---	y			y	y	y		y	y	y	y	y
Action	GET	C18.1	18.2	y			y		y	y	y				y
Action invoke	GET	C19.1	B11 B12 14.4	y			y		y	y	y		y		y
	PUT	C19.2	---	y			y	y	y	y	y	y	y	y	y
	POST	C19.3	---	y	y		y	y	y	y	y	y	y	y	y
Objects	POST	§B9.1	§14.4		y		y			y	y				y

For a given status code, the specific headers and body returned by these resources vary little between the different resources; this is especially so for the failure scenarios (4xx and 5xx).

This section (§C13) describes all the responses irrespective of the resource called. Sections §14 to §C18.2 identify the various request/response scenarios for each of the domain object resources. In each case they define the request URL, headers and body, and also identify the standard (success) response headers and body, if any.

## 13.1 Request succeeded, and generated a representation

For resources that return a body containing some representation.

If the response has been generated by a resource that has also modified the state (e.g. modifying a property or collection or invoking an action), then there will be no self link. This is to discourage clients from bookmarking the link.

---

### 13.1.1 Status code

---

- 200 "OK"

---

### 13.1.2 Headers

---

- **Content-Length:**
  - size of the entity body<sup>30</sup>
- **Content-Type** (objects):
  - application/json;profile=".../xxx";x-ro-domain-type="yyy"
    - where xxx indicates the representation type of either "object" or "action-result" (for an action returning an object); see §2.4.1
    - where yyy indicates the domain type identifier §2.4.2;
      - the domain type id (if simple scheme)
      - URI of domain type (if formal scheme)

---

<sup>30</sup> As defined in <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html>.



- **Content-Type** (lists):
  - application/json;profile=".../xxx";x-ro-element-type="yyy"
    - where xxx indicates the representation type of either "object-collection", "list" and "action-result" (for an action returning a list); see §2.4.1
    - where yyy indicates the element type, §2.4.2):
      - the domain type id (if simple scheme)
      - URI of domain type (if formal scheme)
- **Content-Type** (other):
  - application/json;profile=".../xxx "
    - where xxx indicates any other representation type, see §2.4.1
- Caching headers:
  - TRANSACTIONAL, see §A2.13
    - if the object is transactional
  - NON\_EXPIRING, see §A2.13
    - if the implementation can determine that the returned representation is safe to cache (e.g. the returned objects are immutable reference data)
- **ETag**
  - *digest of timestamp*

---

## 13.1.3 Body (representation)

---

The representation will depend on the resource being requested.

## 13.2 Request succeeded, and generated representation of a new object

---

### 13.2.1 Status code

---

- 201 "OK"

---

### 13.2.2 Headers

---

- **Content-Length:**
  - size of the entity body
- **Content-Type:**
  - application/json;profile=".../object";x-ro-domain-type="yyy"
    - where yyy indicates the domain type (for object representations, §2.4.2)
      - the domain type id (if simple scheme)
      - URI of domain type (if formal scheme)

- Caching headers:
  - TRANSACTIONAL, see §A2.13
    - if the object is transactional
- **ETag**
  - *digest of timestamp*
- **Location:**
  - the URI of the resource of the object just created

---

### 13.2.3 Body (representation)

---

Representation of a domain object, see §14.4.

## 13.3 Request succeeded, but generated no content

This response is most often generated as the result of a validation succeeding (if **x-ro-validate-only** is supported, §A3.2). Note, by contrast, that invoking a void action DOES return a representation §19.4.4.

---

### 13.3.1 Status code

---

- 204 "No content"

---

### 13.3.2 Headers

---

- **Warning** (optional)
  - indicates an informational message generated by the domain object's business logic

---

### 13.3.3 Body

---

- empty

## 13.4 Bad request

Generated either as the result of a syntactically invalid request

---

### 13.4.1 Status code

---

- 400 ("bad request")
  - missing arguments
  - arguments are malformed

---

## 13.4.2 Headers

---

- **Warning**
    - Message text is implementation-specific, but should describe the error condition sufficiently to enable developer-level debugging
- 

## 13.4.3 Body

---

If arguments §A2.9.2/properties (§14.2, §B9.1) are malformed, (for example, incorrect datatype), then the response body is the same as the request body, but additionally will indicate the arguments/properties that are invalid using an **"invalidReason"** json-property to indicate why they are invalid.

For example:

```
{
  "fromDate": {
    "value": "2009-13-33"
    "invalidReason": "could not be parsed as a date"
  }
  ...,
}
```

---

## 13.5 Not authorized (user is not authenticated)

---

### 13.5.1 Status Code

---

- 401 "Forbidden"
- 

### 13.5.2 Headers

---

- **WWW-Authenticate**
    - standard authentication challenge header
- 

### 13.5.3 Body

---

- empty
- 

---

## 13.6 Forbidden (user not authorized to access resource)

---

If the user attempts to invoke a resource that is disabled.

---

### 13.6.1 Status Code

---

- 403 "Forbidden"
-

---

### 13.6.2 Headers

---

- **Warning**
  - same text as "**disabledReason**" in object representation

---

### 13.6.3 Body

---

- empty

## 13.7 Object or object member not found or not visible

This is the response if a requested object or object member does not exist, or if the object/member exists but is not visible based on the current user's credentials.

---

### 13.7.1 Status Code

---

- 404 "Not found"

---

### 13.7.2 Headers

---

- **Warning**
  - No such service {serviceId}
  - No such domain object {oid}
  - No such property {propertyId}
  - No such collection {collectionId}
  - No such action {actionId}

---

### 13.7.3 Body

---

- empty

## 13.8 Resource has invalid semantics for method called

---

### 13.8.1 Status code

---

- 405 ("method not allowed")

---

### 13.8.2 Headers

---

- **Allow**
  - comma-separated list of methods that are supported, as per RFC 2616
- **Warning**
  - object is immutable (if attempt any PUT, DELETE or POST)
  - action is not side-effect free (if attempt GET Act/Invoke)
  - action is not idempotent (if attempt PUT Act/Invoke)
  - collection is not a list (if attempt POST Collection)
  - collection is not a set (if attempt PUT Collection)
  - object cannot be safely deleted (if attempt DELETE Object)

---

### 13.8.3 Body

---

- empty

## 13.9 Not acceptable

The client has specified an **Accept** header that does not include a media type provided by the resource.

For resources that return "application/json" representations, a 406 response code will occur if the **Accept** header is set to "application/json" but has an incompatible "profile" parameter. For example, specifying a profile=".../collection" for anything other than a collection resource § 17.1 will return a 406.

A 406 can also be returned for blob/clob property resources § 16.2.2 when there is a mismatch between the **Accept** header and the media type of the stored blob/clob. For example, setting **Accept** to "image/jpeg" for a "video/h264" will return a 406.

---

### 13.9.1 Status code

---

- 406 ("not acceptable")

---

### 13.9.2 Headers

---

- none

---

### 13.9.3 Body

---

- empty

---

## 13.10 Precondition failed (object changed by other user)

---

---

### 13.10.1 Status code

---

- 412 "precondition failed"

---

### 13.10.2 Headers

---

- **Warning**
  - "Object changed by another user".

The **ETag** header is deliberately *not* returned in order to force client to re-retrieve an up-to-date representation

---

### 13.10.3 Body

---

- empty

---

## 13.11 Unprocessable Entity (validation error)

---

Generated as the result of a validation failure.

---

### 13.11.1 Status code

---

- 422 ("unprocessable entity")
  - property member values are invalid (if updating multiple properties § 14.2, or if persisting a proto-persistent object § B9.1)
  - "Arguments invalid"
    - details are provided in the body

---

### 13.11.2 Headers

---

- **Warning**

- Message text is implementation-specific, but should describe the error condition sufficiently to enable developer-level debugging

---

### 13.11.3 Body

---

If arguments §A2.9.2/properties (§14.2, §B9.1) are invalid, then the response body is the same as the request body, but additionally will indicate the arguments/properties that are invalid using an **"invalidReason"** json-property to indicate why they are invalid

For example:

```
{
  "fromDate": {
    "value": "2009-12-01"
    "invalidReason": "The from date cannot be in the past"
  }
  ...,
}
```

If no individual argument/property was invalid, but the set of such is invalid (e.g. fromDate > toDate), then an **"x-ro-invalidReason"** json-property is provided at the root of the map.

For example:

```
{
  "fromDate": ...,
  "toDate": ...,
  "x-ro-invalidReason": "To date cannot be before from date"
}
```

The json-property has the prefix **"x-ro-"** in this case in order to avoid clashes with the argument/property names

## 13.12 Precondition header missing

This represents a syntax error where a required precondition header (for example, **If-Match** if modifying state of a resource) was not included in the request.

---

### 13.12.1 Status code

---

- 428 "precondition header missing"

---

### 13.12.2 Headers

---

- **Warning**
  - "If-Match header required with last-known value of ETag for the resource in order to modify its state".

---

## 13.12.3 Body

---

- empty

## 13.13 Domain logic failed, or Implementation defect

---

### 13.13.1 Status code

---

- 500 ("internal server error")

---

### 13.13.2 Headers

---

- **Warning**
  - error message raised by business logic in the domain model, or
  - exception message raised by the Restful Objects implementation itself

---

### 13.13.3 Body

---

- the error representation §B10.



## 14 DOMAIN OBJECT RESOURCE & REPRESENTATION

The domain object resource can be used to obtain the summary domain object representation § 14.4 for a particular domain object, and can also be used to update or delete individual persisted domain object instances (where allowed).

The endpoint URL (using templated URI<sup>31</sup> syntax) for this resource is:

```
/objects/{domainType}/{instanceId}
```

where:

- {domainType} uniquely identifies the object's domain type, and
- {instanceId} uniquely identifies an object instance of that type

Together the {domainType}/{instanceId} may be referred to as the object identifier, or oid. The specification does not mandate the format of either the domainType or the instanceId; the following are all potentially valid forms:

```
/objects/customers/123
/objects/myApp.Customer/123
/objects/CUS/123
/objects/ORD/123-456
/objects/countries/USA
```

### 14.1 HTTP GET

Obtain a summary representation of a domain object § 14.4. The intention is that enough information is provided to render the object in the client's UI.

---

#### 14.1.1 Request

---

##### 14.1.1.1 Query String

- **x-ro-domain-model** (optional, § A3.1)
  - "simple"
  - "formal"

##### 14.1.1.2 Headers

- **Accept**
  - application/json
  - application/json;profile=".../object"

---

<sup>31</sup> <http://tools.ietf.org/html/draft-gregorio-uritemplate>

If the "profile" parameter is specified to any value other than ".../object", then a 406 response code should be returned §13.9.

### 14.1.1.3 **Body**

- N/A

---

## 14.1.2 **Success Response**

---

As per §13.1 (200), returning a domain object representation §14.4.

## 14.2 **HTTP PUT**

Update multiple properties of the object at the same time, or alternatively validate the proposed values but do not modify the object.

---

### 14.2.1 **Request**

---

The request can either be to update the properties, or to request validation of the proposed values using the **x-ro-validate-only** query param §A3.2.

#### 14.2.1.1 **Query String**

- None

#### 14.2.1.2 **Headers**

- **If-Match**
  - *timestamp digest*
    - obtained from **ETag** header of representation

#### 14.2.1.3 **Body**

The body is the map of new properties, as per §A2.9.2.3. Note that any blob/clob properties must be in-lined within this map. (Contrast this to updating of individual properties where the value of a blob/clob can be PUT in its native media type, §16.2).

In addition, it may include the reserved query parameters:

- **x-ro-domain-model** (optional, §A3.1)
  - "simple"
  - "formal"
- **x-ro-validate-only** (optional, §A3.2)
  - "true"
    - only validate the request, do not modify the property

For example:

```
{
  "firstName": {
    "value": ...
  },
  "lastName": {
    "value": ...
  },
  ...
}
```

---

## 14.2.2 Success Response

---

As per 13.3 (200), returning a domain object representation §14.4. Because the resource has mutated the state, there will be no self link (so that it cannot be bookmarked by clients).

## 14.3 HTTP DELETE

Deletes an object. This is an optional capability §B8 because implementing a generic 'delete object' capability - which includes managing any references to the deleted object throughout the system - is potentially complex, and not necessarily practicable for many implementations.

If the implementation does support the capability then it must also determine that it is safe to delete the object. A **405** ("method not allowed") error will be returned otherwise.

---

### 14.3.1 DELETE Request

---

#### 14.3.1.1 Query String

- none

#### 14.3.1.2 Headers

- **If-Match**
  - *timestamp digest*
    - obtained from **ETag** header of representation

#### 14.3.1.3 Body

- N/A

---

### 14.3.2 DELETE Success Response

---

As per §13.3 (204), returning no representation.

## 14.4 Representation

The domain object representation provides summary information about a single domain object instance, along with links to other sub-resources by which the domain object may be interacted with, or mutated. As such, it is the single most important representation defined by Restful Objects.

The **Content-Type** for the representation is:

```
application/json;profile=".../object";x-ro-domain-type="yyy"
```

where yyy identifies the domain type identifier:

- the domain type id of the returned object (simple scheme)
- the URI to the domain type of the returned object (formal scheme)

The representation is typically generated from the Domain Object resource § 14.1, though it can also be generated by the Domain Service resource § 15 (since Restful Objects regards a domain service as being just a well-known domain object). It may also be obtained as the result of updating multiple properties § 14.2, or of persisting a proto-persistent object § B9.

The links from the domain object representation to other resources are as shown in the diagram below:

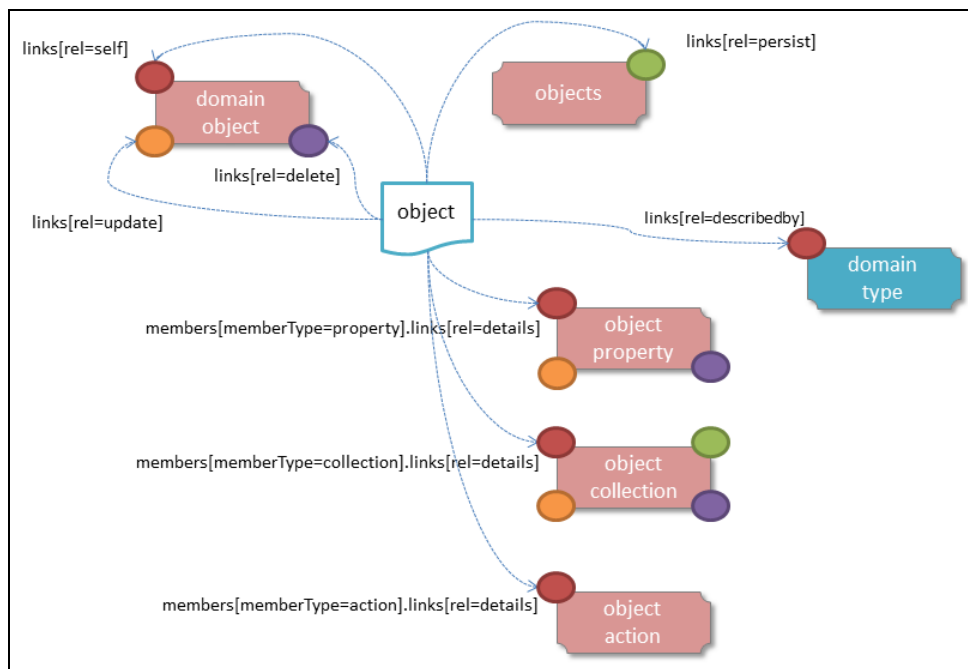


FIGURE 8: DOMAIN OBJECT REPRESENTATION

# Restful Objects

For example, the representation of a (persistent domain entity) Order might be:

```
{
  "domainType": "ORD",
  "instanceId": " 123",
  "title": "Joe Blogg's Order #1",
  "members": {
    ...
  },
  "links": [ {
    "rel": "self",
    "href": "http://~/objects/ORD/123",
    "type": "application/json;profile=\"../object\"",
    "method": "GET",
  },
  ...
],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
<b>links</b>	list of links to other resources.
<b>links[rel=self]</b>	(optional); link to a resource that can obtain this representation. Note that the href for a service will be http://~/services/{serviceId}. Discussed further below.
<b>domainType</b>	(optional) the domain type to use when building template URIs. Discussed further below
<b>instanceId</b>	(optional) the instance identifier, to use when building template URIs. Discussed further below.
<b>serviceId</b>	(optional) the service Id. Present only if the object is a domain service §15.
<b>title</b>	a string identifier of the object, suitable for rendering in a UI.
<b>members</b>	map of object members (properties, collections, actions)
<b>links[rel=.../persist]</b>	(optional) persist the (proto-persistent) domain object. Discussed further below.
<b>links[rel=.../update]</b>	(optional) link to modify multiple properties of the domain object (using §14.2). The link is present only for persistent domain entities that have at least one modifiable property. Discussed further below.
<b>links[rel=.../delete]</b>	(optional) delete the (persistent) domain object. Discussed further below.
<b>links[rel=icon]</b>	(optional) link to an image representing a scalable icon for this object
<b>extensions</b>	additional information about the resource.

### *"domainType"*

The **"domainType"** json-property is only present for the simple scheme §A3.1.1; if the formal scheme §A3.1.2 is supported then the **"domainType"** can be obtained from the domain-type representation §D22.2.

Domain services do *not* have a "domainType" json-property.

### *"instanceId" , "serviceId" and "links[rel=self]"*

The **"instanceId"** json-properties is present for persistent domain entities and for addressable view models §A2.2, and can (with the **"domainType"** json-property) be used to construct URLs to other resources for the domain object as required.

Proto-persistent domain objects and (non-addressable) view models §A2.2 do not have an **"instanceId"** because they do not correspond to any server-side state that can be directly addressed; nor do they have a 'self' link, for the same reasons. The **"serviceId"** json-property performs much the same function as **"instanceId"**, allowing the URL for domain services to be constructed. The **"serviceId"** is present only for domain services. Domain services do not have a **"instanceId"** json-property.

### *"members"*

The **"members"** map contains an entry for every (visible) member. It is described in more detail in the sections below §14.4.1, §14.4.2, §14.4.3<sup>32</sup>.

### *"links[rel=.../update]"*

For persistent domain objects, there may optionally be a **rel=".../update"** link to update all properties of the domain object.

This link is not guaranteed to be present, however; if none of the properties of an object are updatable then the update properties link will not be present.

Also, proto-persistent domain objects and view models will never have an update link.

### *"links[rel=.../delete]"*

For persistent domain objects, there may optionally be a **rel=".../delete"** link to delete the domain object.

This link is not guaranteed to be present, however. Support for deleting objects is an optional capability §B8.2, and so is not guaranteed to be supported by every framework implementation. If it is supported, then the implementation should define its own mechanism to restrict which objects can be deleted, and which may not.

---

<sup>32</sup> The reserved **x-ro-follow-links** query parameter may also be used to request that more detailed information is returned in the representation.

## Restful Objects

---

Also, proto-persistent domain objects and view models will never have a delete link.

*"links[rel=.../persist]"*

For proto-persistent domain objects, a **rel**=".../persist" link is provided.

The arguments map for this link is a subset of the object representation itself, containing the (property) members of the domain object itself, along with the domain type (specified either with the simple scheme or formal scheme). In particular, the **"members"** map need only specify id (as the key) and the **"value"** of each property member. There is no need to specify other information (such as the **"memberType"**).

For example, the persist link for an Order using the simple metadata scheme might look like:

```
"links": [
  {
    "rel": ".../persist",
    "href": "http://~/objects",
    "type": "application/json;profile=\".../object\"",
    "method": "POST",
    "arguments": {
      "domainType": "x.Order",
      "members": {
        "placedBy": {
          "value": ...
        },
        "placedOn": {
          "value": ...
        },
        ...
      }
    }
  },
  ...
]
```

If the formal scheme is used, then the link might look like:

```
"links": [
  {
    "rel": ".../persist",
    "href": "http://~/objects",
    "type": "application/json;profile=\".../object\"",
    "method": "POST",
    "arguments": {
      "links": [ {
        "rel": "describedby",
        "href": "http://~/domain-types/x.Order",
        ...
      }
    ],
    "members": {
      "placedBy": {
        "value": ...
      },
      "placedOn": {
        "value": ...
      },
    },
  },
  ...
]
```

```
    ...  
  }  
}  
]
```

Note that the **"members"** map must provide values for every property of the domain object, not just those that are visible.

## *"links" and "extensions"*

Domain model information about the type is available through either the **"links"** or the **"extensions"** json-properties. This is discussed separately in §14.4.4.

Implementations are free to add to their own links/properties to **"links"** and **"extensions"** as they require.

### 14.4.1 Properties

The **"members"** map contains an entry for every (visible) property. This entry contains a subset of the information shown in the detailed property representation §16.4. The intention is to provide enough information to render the property value in a user interface without having to make additional requests.

For example, the "createdOn" property would look something like:

```
"members": {  
  "createdOn": {  
    "memberType": "property",  
    "value": ...,  
    "disabledReason": ...,  
    "links": [ {  
      "rel": ".../details;property=\"createdOn\"",  
      "href": "http://~/objects/ORD/123/properties/createdOn",  
      "type": "application/json;profile=\".../object-property\"",  
      "method": "GET"  
    },  
    ...  
  ],  
  "extensions": { ... }  
},  
...  
}
```

where the member's id ("createdOn" in the example above) is used as a unique key in the **"members"** map, and its value being the following map:

JSON-Property	Description
<b>memberType</b>	the constant value "property"
<b>value</b>	(optional) the current value of the property, either a scalar, a (link representing a) reference, or null. Discussed further below.



## Restful Objects

JSON-Property	Description
<b>disabledReason</b>	(optional) if populated then indicates the reason why the property cannot be modified.
<b>links</b>	list of links to resources.
<b>links[rel=.../details]</b>	(optional) link to the detailed representation of the property, §16.4 (e.g. to access defaults and choices).
<b>links[rel=.../attachment]</b>	(optional) link to the property value if it is an attachment. Discussed further below
<b>extensions</b>	map of additional information about the resource.

### *"value" and "links[rel=.../attachment;...]"*

The "value" json-property holds the in-lined value of the property, though depending on the nature of the domain object and the type of the property, it may or may not be present:

- if the property value is null, then the "**value**" json-property will be present and set to the JSON null value
- for proto-persistent domain objects and (non-addressable) view models (§A2.2), the "**value**" is always present.
- for persistent domain objects and addressable view models (with server-side state §A2.2), the "**value**" is always present for non-blobs/clobs §A2.5
- for blobs/clobs in implementations that do not support attachments §A-52, again the "**value**" is present
- however, for persistent domain objects which support attachments the "**value**" is omitted. Instead a link to the attachment will be available. This link serves up the property value directly with the correct media type (e.g. as an image/jpg).

From the client's perspective, this means that there is always either a "**value**" json-property or a "**links[rel=.../attachment;...]**" json-property.

### *"links" and "extensions"*

Other domain model information about the property is available through either the "**links**" or the "**extensions**" json-properties. The information provided through these json-properties is the same as provided in the domain object property representation, see §16.4.3.

Implementations are free to add to their own links/json-properties to "**links**" and "**extensions**" as they require

---

## 14.4.2 Collections

---

The "**members**" map also contains an entry for every (visible) collection, which provides a link to the corresponding Object Collection resource.

## Restful Objects

The member entry may also provide summary information about the collection (for example, its size) so that the client can render the collection without having to make additional requests to the server.

However, if the domain object being represented has no corresponding server-side state (§A2.2), then the collection's representation also in-lines the collection representation §17.5.

As for (object) properties, the json-property representing a collection has a type, a details link, and links to the state.

For example, the Order's items collection would look something like:

```
"members": {  
  ...,  
  "items": {  
    "memberType": "collection",  
    "disabledReason": ...,  
    "value": [ ... ],  
    "size": ...,  
    "links": [ {  
      "rel": ".../details;collection=\"items\"",  
      "href": "http://~/objects/ORD/123/collections/items",  
      "type": "application/json;profile=\".../object-collection\"",  
      "method": "GET"  
    }, ... ],  
    "extensions": { ... }  
  },  
  ...  
}
```

where the member's id is used as a unique key in the "**members**" map, and its value being the following map:

JSON-Property	Description
<b>memberType</b>	the constant value "collection"
<b>disabledReason</b>	(optional) if populated then indicates the reason why it is not possible to add to or remove from the collection.
<b>value</b>	(optional) contains a representation of the contents of the collection. Discussed further below.
<b>size</b>	(optional) contains a count of the elements in the collection. Discussed further below.
<b>links</b>	links to other resources.
<b>links[rel=.../details]</b>	(optional) link to the detailed representation of the collection, §17.5, which includes such information as defaults and choices. Discussed further below.
<b>extensions</b>	additional information about the resource.

### *"links[rel=.../details]", "value" and "size"*

As noted above, representations of domain objects without corresponding server-side state (§A2.2) will in-line the **"value"** of the collection. For these domain objects, there is no "size" json-property and there is no **"links[rel=.../details;...]"** link.

Domain objects with server-side state, however, need not provide a **"value"**. Instead, they may provide a **"links[rel=.../details]"** which when followed will return the value in the collection's detailed representation §17.5.

This behaviour allows implementations to load only the object and not all of its related references (in other words, lazy loading).

### *"links" and "extensions"*

Other domain model information about the collection is available through either the **"links"** or the **"extensions"** json-properties. The information provided through these json-properties is the same as provided in the domain object collection representation, see §17.5.3.

Implementations are free to add to their own links/json-properties to **"links"** and **"extensions"** as they require

---

### 14.4.3 Actions

---

The **"members"** map also contains an entry for every (visible) action. Note however that only domain objects with corresponding server-side state (§A2.2) will have actions.

The information provided is a subset of the information shown in the detailed action representation §18.2 (obtainable from the GET Action resource §18.2). The intention is to provide enough information to render the action without having to make additional requests.

Like a property or a collection, an action has a link to 'details' which allows additional information (specifically, choices and defaults on parameters) to be obtained that might otherwise be expensive to compute. It also includes a link to follow in order to invoke the action.

## Restful Objects

---

For example, the Order's submit() action might be represented as:

```
"members": {  
  ...  
  "submit": {  
    "memberType": "action",  
    "disabledReason": ...,  
    "links": [ {  
      "rel": ".../details;action=\"submit\"",  
      "href": "http://~/objects/ORD/101/actions/submit",  
      "type": "application/json;profile=\".../object-action\"",  
      "method": "GET"  
    } ... ],  
    "extensions": { ... }  
  },  
  ...  
}
```

where the member's id is used as a unique key in the "**members**" map, and its value being the following map:

JSON-Property	Description
<b>memberType</b>	the constant value "action"
<b>disabledReason</b>	(optional) if populated then indicates the reason why the action may not be invoked.
<b>links</b>	list of links to other resources.
<b>links[rel=.../details]</b>	link to the detailed representation of the action, §18.2.
<b>extensions</b>	additional metadata about the resource

### *"links" and "extensions"*

Other domain model information about the action is available through either the "**links**" or the "**extensions**" json-properties. The information provided through these json-properties is the same as provided in the domain object action representation, see §18.2.3.

Restful Objects defines no further standard links/json-properties for "**links**" or "**extensions**". However, implementations are free to add to their own links/json-properties as they require.

---

#### 14.4.4 Domain model information

---

Domain model information is available through either the "**links**" or the "**extensions**" json-properties.

## *Simple scheme*

Implementations that support the *simple* scheme provide extra data in the **"extensions"** json-properties. For example:

```
"extensions": {
  "domainType": "x.Order",
  "friendlyName": "Order",
  "pluralName": "Orders",
  "description": "An order that has been placed by a customer",
  "isService": false
  "memberOrder": 1
}
```

See §A3.1.1 for the full definitions of these json-properties.

## *Formal scheme*

Implementations that support the *formal* scheme §A3.1.2 provide an additional link in the **"links"** json-property:

```
"links": [
  {
    "rel": "describedby",
    "href": "http://~/domain-types/x.Order",
    "type": "application/json;profile=\"../domain-type\"",
    "method": "GET"
  },
  ...
]
```

which links to the domain type resource §D22 corresponding to this domain object.



# 15 DOMAIN SERVICE RESOURCE

A domain service is a well-known (singleton) domain object that typically acts as a repository and/or factory, providing actions for obtaining other domain object instances (although services can provide any kind of functionality in relation to objects).

Domain services can be accessed from the object list representation §B11 returned by the GET Domain Services resource §B7.

## 15.1 HTTP GET

Obtain summary representation of a domain service..

The endpoint URL for this resource is:

`/services/{serviceId}`

where:

- `{serviceId}` is a unique identifier for the service

---

### 15.1.1 Request

#### 15.1.1.1 Query String

- **x-ro-domain-model** (optional, §A3.1)
  - "simple"
  - "formal"

#### 15.1.1.2 Headers

- **Accept**
  - `application/json`
  - `application/json;profile=".../object"`

#### 15.1.1.3 Body

- N/A

---

### 15.1.2 Success Response

As per §13.1 (200), returning a domain object representation §14.4.

## 16 PROPERTY RESOURCE & REPRESENTATION

The (domain object) property resource can be used to obtain the detailed domain object property representation § 16.4 for a particular domain object instance. It also allows the value of that property to be modified (or to validate a proposed new value for a property).

The endpoint URL for this resource is:

`/objects/{domainType}/{instanceId}/properties/{propertyId}`

where:

- `{domainType}` uniquely identifies the object's type, and
- `{instanceId}` uniquely identifies an object instance of that type
- `{propertyId}` is the property identifier

### 16.1 HTTP GET

Obtain a detailed representation of a property § 16.4.

This resource is typically requested as a result of following a link from the domain object representation § 14.4.

---

#### 16.1.1 Request (any property type)

---

To return a representation § 16.4 of the property; if a non-blobClob then this will include the property's current value, if a blob/clob then will include a link `rel=".../attachment"` by which the blob/clob can be retrieved, § 16.1.2.

##### 16.1.1.1 Query String

- **x-ro-domain-model** (optional, § A3.1)
  - "simple"
  - "formal"

##### 16.1.1.2 Headers

- **Accept**
  - `application/json`
  - `application/json;profile=".../object-property"`

##### 16.1.1.3 Body

- N/A



---

## 16.1.2 Request (for blobClob attachment)

---

If the property is a blob/clob, then getting the property with an "application/json" **Content-Type** §16.1.1 will return a representation of the property §16.4 that include an **rel=".../attachment"** link. This link refers back to the same resource URL, but indicates the media type<sup>33</sup> for the **Accept** header in order to obtain the actual blob/clob.

### 16.1.2.1 Query String

- None

### 16.1.2.2 Headers

- **Accept**
  - as specified in the rel=.../attachment link
  - eg image/pdf, image/jpeg, video/h264

### 16.1.2.3 Body

- Blob/clob resource

---

## 16.1.3 Success Response

---

As per §13.1 (200), returning an object property representation §16.4.

## 16.2 HTTP PUT

Update property value of an object, or check that the proposed new value for the property would be valid without making the change. The PUT method may be used to clear the property by passing in a null value, but the recommended practice is to use DELETE §C16.3 for that purpose.

Properties that are blob/clobs support two different request formats. Non blob/clobs support only a single request format.

---

## 16.2.1 Request (for non-blobClobs)

---

Updating non-blobClob properties is performed by PUTting the value in-lined within a JSON map §A2.9.2.2.

### 16.2.1.1 Query String

- None

### 16.2.1.2 Headers

- **Content-Type:** application/json

---

<sup>33</sup> <http://www.iana.org/assignments/media-types/index.html>

- **If-Match**
  - *timestamp digest*
    - obtained from **ETag** header of representation
    - only validate the request, do not modify the property

### 16.2.1.3 Body

- should be formatted as a single argument node §A2.9.2.2.

In addition, it may include the reserved query parameters:

- **x-ro-domain-model** (optional, §A3.1)
  - "simple"
  - "formal"
- **x-ro-validate-only** (optional, §A3.2)
  - "true"
    - only validate the request, do not modify the property

---

## 16.2.2 Request (if blobClobs)

---

Updating blobClob properties is performed by PUTting the actual value (e.g. image), with appropriate content type.

Note that optional validation (**x-ro-validate-only**) and domain type metadata preferences (**x-ro-domain-model**) are not supported for blobClobs.

### 16.2.2.1 Query String

- none

### 16.2.2.2 Headers

- **Content-Type:** (depends on property type)
  - eg image/jpeg, image/png, application/pdf
- **If-Match**
  - *timestamp digest*
    - obtained from **ETag** header of representation
    - only validate the request, do not modify the property

### 16.2.2.3 Body

- a byte array (for blobs)
- a character array (for clobs)

---

## 16.2.3 Success Response

---

As per §13.1 (200), returning an object property representation §16.4.

## 16.3 HTTP DELETE

This is the recommended resource for clearing a property value, or for validating that a property can be cleared but without making the change.

Strictly speaking the DELETE Object Property resource is redundant because it is also possible to clear a property using the PUT method, passing in a null value. However, the DELETE Object Property resource has been included in the spec because it offers a simpler syntax (no body to pass in) and because it is more 'intentional' (the intent of calling the resource is clearer to anyone reading the code).

---

### 16.3.1 Request

#### 16.3.1.1 Query Params

- None

#### 16.3.1.2 Headers

- **If-Match**
  - *timestamp digest*
    - obtained from **ETag** header of representation
    - only validate the request, do not modify the property

#### 16.3.1.3 Body

- **x-ro-domain-model** (optional, §A3.1)
  - "simple"
  - "formal"
- **x-ro-validate-only** (optional, §A3.2)
  - "true"
    - only validate the request, do not modify the property

---

### 16.3.2 Success Response

As per §13.1 (200), returning an object property representation §16.4. Because the resource has mutated the state, there will be no self link (so that it cannot be bookmarked by clients).

## 16.4 Representation

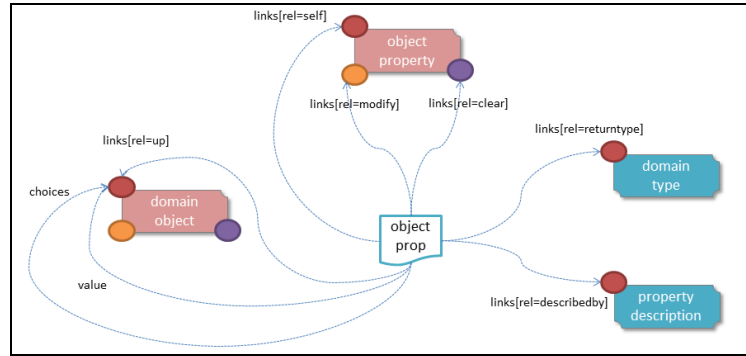
The domain object property representation provides full details about a property of a domain object instance, and provides links to resources to allow the property to be modified (if it is not disabled).

# Restful Objects

The **Content-Type** for the representation is:

`application/json;profile="../../../object-property"`

The links from the object property representation to other resources are as shown in the diagram below:



**FIGURE 9: OBJECT PROPERTY REPRESENTATION**

For example, the representation of an Order's deliveryOption property might be:

```
{
  "deliveryOption": {
    "disabledReason": ...,
    "value": ...,
    "choices": [ ... ]
    "links": [ {
      "rel": "self",
      "href": "http://~/objects/ORD/123/properties/deliveryOption",
      "type": "application/json;profile="../../../object-property\"",
      "method": "GET",
    },
    {
      "rel": ".../modify;property=\"deliveryOption\"",
      ...
    },
    {
      "rel": ".../clear;property=\"deliveryOption\"",
      ...
    },
    {
      "rel": "up",
      ...
    },
    ...
  ],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
links	list of links to resources.
links[rel=self]	link to a resource that can obtain this representation
id	property ID, to use when building templated URIs

# Restful Objects

JSON-Property	Description
<b>value</b>	(optional) the current value of the (non blob/clob) property, §16.4.1.
<b>choices</b>	(optional) list of suggested/recommended choices for the (non blob/clob) property, §16.4.1.
<b>disabledReason</b>	(optional) if populated then indicates the reason why the property cannot be modified.
<b>links[rel=.../modify]</b>	(optional) link back to self to modify property value; discussed below, §16.4.2.
<b>links[rel=.../clear]</b>	(optional) link back to self to clear property value; discussed below, §16.4.2.
<b>links[rel=up]</b>	link to the object that is the owner of this property.
<b>links[rel=.../attachment]</b>	(optional) link to resource returning property if a blob/clob, §16.4.1.
<b>extensions</b>	additional information about the resource.

## *"choices"*

The **"choices"** json-property lists a set of values which are valid for the property. (It is up to the implementation to determine whether this set of choices is exclusive ( i.e. whether other values may also be valid) or not.

## *"links" and "extensions"*

Both the **"links"** and the **"extensions"** json-properties may contain domain model information; this is discussed in §16.4.3.

Restful Objects defines no further standard child properties for the **"extensions"** json-property. Implementations are free to add further links/json-properties to **"links"** and **"extensions"** as they require.

---

## 16.4.1 Property values and choices

---

For value properties (other than blobs/clobs), the **"value"** and **"choices"** json-properties are directly parseable strings:

```
{
  ...
  "id": "deliveryOptions",
  ...
  "value": "PRIORITY",
  "choices": ["PRIORITY", "STANDARD", "PARCEL"],
  ...
}
```

## Restful Objects

---

For reference properties, the **"value"** and **"choices"** json-properties hold links to other object resources:

```
{
  "id": "paymentMethod",
  "...": {
    "value": {
      "rel": ".../value;property=\"paymentMethod\"",
      "href": "http://~/objects/PMT/VISA",
      "type": "application/json;profile=\".../object\"",
      "method": "GET",
      "title": "Visa"
    },
    "choices": [
      {
        "rel": ".../choice;property=\"paymentMethod\"",
        "href": "http://~/objects/PMT/VISA",
        "type": "application/json;profile=\".../object\"",
        "method": "GET",
        "title": "Visa"
      },
      {
        "rel": ".../choice;property=\"paymentMethod\"",
        "href": "http://~/objects/PMT/AMEX",
        "type": "application/json;profile=\".../object\"",
        "method": "GET",
        "title": "American Express"
      },
      {
        "rel": ".../choice;property=\"paymentMethod\"",
        "href": "http://~/objects/PMT/MCRD",
        "type": "application/json;profile=\".../object\"",
        "method": "GET",
        "title": "Mastercard"
      }
    ]
  }
}
```

For blob/clob value properties, the **"value"** json-property is omitted. Instead a **link[rel=".../attachment"]** json-property provides a link that can be followed, with the appropriate Accept header, to obtain the blob/clob:

```
{
  "id": "scannedSignature",
  "...": {
    "links": [
      {
        "rel": ".../attachment;property=\"scannedSignature\"",
        "href": "http://~/objects/CUS/123/property/scannedSignature",
        "type": "image/jpeg",
        "method": "GET"
      }
    ]
  }
}
```

If the property is null, then there will be neither a **"value"** nor a **link[rel=".../attachment"]** json-property.

## 16.4.2 Property modification

If the property is modifiable, then the **"modify"** and **"clear"** json-properties provide links to the resources used to change the property's state.

For example:

```
{
  "id": "deliveryTime",
  "links": [ {
    "rel": ".../modify;property=\"deliveryTime\"",
    "href": "http://~/objects/ORD/123/properties/deliveryTime",
    "type": "application/json;profile=\".../object-property\"",
    "method": "PUT",
    "arguments": {
      "value": null
    }
  }, {
    "rel": ".../clear;property=\"deliveryTime\"",
    "href": "http://~/objects/ORD/123/properties/deliveryTime",
    "type": "application/json;profile=\".../object-property\"",
    "method": "DELETE"
  },
  ...
]
```

where:

JSON-Property	Description
<code>links[rel=.../modify]</code>	link back to self to modify property value; not included if the property is disabled
<code>links[rel=.../clear]</code>	link back to self to clear property value; not included if the property is disabled

The new value (for the **"modify"**) is sent in the body request via HTTP PUT. Validation of properties occurs when the modify is made. If only validation is of a property is required, then specify the **x-ro-validate-only** request parameter §A3.2.

If the domain object property is NOT modifiable, then the representation will include a **"disabledReason"** json-property that indicates the reason (or just the literal "disabled") why the value of the property cannot be modified:

```
{
  "disabledReason":
    "Cannot add items to order that has already shipped",
  ...
}
```

where:

JSON-Property	Description
<b>disabledReason</b>	indicates the reason why the property cannot be modified/cleared; only included if the property is disabled.

---

## 16.4.3 Domain model information

---

Domain model information is available through either the **"links"** or the **"extensions"** json-properties.

### 16.4.3.1 Simple scheme

Implementations that support the *simple* scheme provide extra data in the **"extensions"** json-property. For example:

```
"extensions": {
  "friendlyName": "Delivery Time",
  "description": "Time that the order will be delivered",
  "returnType": ...
  "optional": false,
  "format": ... // for string properties only
  "maxLength": ... // for string properties only
  "pattern": ... // for string properties only
  "memberOrder": 3
}
```

See §A3.1.1 for the full definitions of these json-properties.

### 16.4.3.2 Formal scheme

Implementations that support the *formal* scheme §A3.1.2 provide an additional link only in the **"links"** json-property:

```
"links": [
  {
    "rel": "describedby",
    "href":
      "http://~/domain-types/x.Order/properties/deliveryTime",
    "type":
      "application/json;profile=\".../ property-description\"",
    "method": "GET"
  }
]
```

which links to the domain property description resource §D22.2 corresponding to this domain object property.



## 17 COLLECTION RESOURCE & REPRESENTATION

The (domain object) collection resource can be used to obtain the detailed domain object collection representation §17.5 for a particular domain object instance. It also allows elements to be added to, or removed from, the collection (or optionally to validate the adding and removal of elements without modifying the collection).

The endpoint URL for this resource is:

`/objects/{domainType}/{instanceId}/collections/{collectionId}`

where:

- {domainType} uniquely identifies the object's type, and
- {instanceId} uniquely identifies an object instance of that type
- {collectionId} is the collection identifier

### 17.1 HTTP GET

Obtain a detailed representation of a collection §17.5.

This resource is typically requested as a result of following a link from the domain object representation §14.4.

---

#### 17.1.1 Request

##### 17.1.1.1 Query String

- **x-ro-domain-model** (optional, §A3.1)
  - "simple"
  - "formal"

##### 17.1.1.2 Headers

- **Accept**
  - application/json
  - application/json;profile=".../object-collection"

##### 17.1.1.3 Body

- N/A

---

#### 17.1.2 Success Response

As per §13.1 (200), returning an object collection representation §17.5. Note that the **Content-Type** will include an "x-ro-element-type" parameter.

## 17.2 HTTP PUT

Add an object to a collection, or alternatively validate that the proposed object may be added to the collection is valid but without making the change.

This method is valid only if the collection has Set semantics (the most common case, where duplicate entries are not permitted).

---

### 17.2.1 Request

#### 17.2.1.1 Query String

- **x-ro-domain-model** (if domain metadata is "selectable", §A3.1)
  - "simple"
  - "formal"

#### 17.2.1.2 Headers

- **If-Match**
  - *timestamp digest*
    - obtained from **ETag** header of representation

#### 17.2.1.3 Body

- should be formatted as a single argument node §A2.9.2.2.

In addition:

- **x-ro-domain-model** (optional, §A3.1)
  - "simple"
  - "formal"
- **x-ro-validate-only** (optional, §A3.2)
  - "true"
    - only validate the request, do not modify the collection

---

### 17.2.2 Success Response

As per §13.1 (200), returning an object collection representation §17.5. Note that the **Content-Type** will include an "x-ro-element-type" parameter.

## 17.3 HTTP POST

Add an object to a collection, or alternatively validate that the proposed object to add to the collection is valid but do not modify the collection.

This method is valid only if the collection has List semantics (where duplicate entries are permitted).

---

## 17.3.1 Request

---

### 17.3.1.1 Query String

- None

### 17.3.1.2 Headers

- **If-Match**
  - *timestamp digest*
    - obtained from **ETag** header of representation

### 17.3.1.3 Body

- should be formatted as a single argument node §A2.9.2.2.

In addition:

- **x-ro-domain-model** (optional, §A3.1)
  - "simple"
  - "formal"
- **x-ro-validate-only** (optional, §A3.2)
  - "true"
    - only validate the request, do not modify the collection

---

## 17.3.2 Success Response

---

As per §13.1 (200), returning an object collection representation §17.5.  
Note that the **Content-Type** will include an "x-ro-element-type" parameter.

---

## 17.4 HTTP DELETE

---

Remove an object from a collection, or validate that an object may be removed from the collection but without making the change.

---

## 17.4.1 Request

---

### 17.4.1.1 Query String

A single query argument should be formatted as a single argument node §A2.9.2.2 referencing the object to remove:

```
{
  "value": {
    "href": "http://~/objects/xxx/yyyy"
  }
}
```

In addition:

- **x-ro-domain-model** (optional, §A3.1)
  - "simple"
  - "formal"
- **x-ro-validate-only** (optional, §A3.2)
  - "true"
    - only validate the request, do not modify the collection

### 17.4.1.2 Headers

- **If-Match**
  - *timestamp digest*
    - obtained from **ETag** header of representation

### 17.4.1.3 Body

- None

---

## 17.4.2 Success Response

---

As per §13.1 (200), returning an object collection representation §17.5. Because the resource has mutated the state, there will be no self link (so that it cannot be bookmarked by clients).

## 17.5 Representation

The domain object collection representation provides full details of a collection of a domain object, and provides links to resources that can modify the contents of the collection, if allowable.

The **Content-Type** for the representation is:

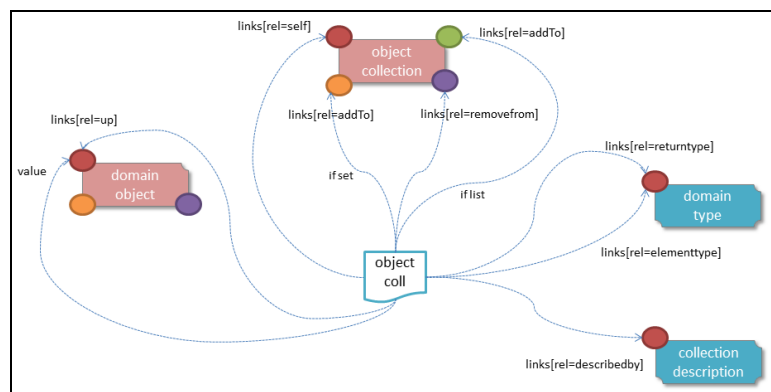
```
application/json;  
  profile=".../object-collection;  
  x-ro-element-type=yyy"
```

where yyy indicates the domain type:

- the domain type id (if simple scheme)
- URI of domain type (if formal scheme)

## Restful Objects

The links from the object collection representation to other resources are as shown in the diagram below:



### FIGURE 10: OBJECT COLLECTION REPRESENTATION

For example, the representation of an Order's items collection might be:

```
{
  "id": "items",
  "value": [ ... ],
  "disabledReason": ...,
  "links": [ {
    "rel": "self",
    "href": "http://~/objects/ORD/123/collections/items",
    "type": "application/json;profile=\"../object-collection\"",
    "method": "GET",
  }, {
    "rel": ".../addTo;collection=\"items\"",
    ...
  }, {
    "rel": ".../removeFrom;collection=\"items\"",
    ...
  }, {
    "rel": "up",
    ...
  }
  ...
],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
links	list of links to other resources.
links[rel=self]	link to a resource that can obtain this representation
id	collection ID, to use when building templated URIs
value	list of links to the domain objects referenced by the collection, §17.5.1.
disabledReason	(optional) if populated then indicates the reason why the collection cannot be modified.

# Restful Objects

JSON-Property	Description
<code>links[rel=.../add-to]</code>	(optional) link back to self to add item to collection; discussed below, §17.5.2.
<code>links[rel=.../remove-from]</code>	(optional) link back to self to remove item from collection; discussed below, §17.5.2.
<code>links[rel=up]</code>	link to the object that is the owner of this collection.
<code>extensions</code>	additional information about the resource.

Both the "**links**" and the "**extensions**" json-properties may contain domain model information; this is discussed in §17.5.3.

Restful Objects defines no further standard child properties for the "**extensions**" json-property. Implementations are free to add further links/json-properties to "**links**" and "**extensions**" as they require.

---

## 17.5.1 Collection values

---

The value of a collection is a list of links to other objects e.g.:

```
"value": [  
  {  
    "rel": ".../value;collection=\"items\"",  
    "href": "http://~/objects/ORI/123-1",  
    "type": "application/json;profile=\".../object\"",  
    "method": "GET",  
    "title": "Harry Potter and the Goblet of Fire"  
  },  
  {  
    "rel": ".../value;collection=\"items\"",  
    "href": "http://~/objects/ORI/123-2",  
    "type": "application/json;profile=\".../object\"",  
    "method": "GET",  
    "title": "Rubiks Cube"  
  },  
  {  
    "rel": ".../value;collection=\"items\"",  
    "href": "http://~/objects/ORI/123-3",  
    "type": "application/json;profile=\".../object\"",  
    "method": "GET",  
    "title": "Xbox"  
  }  
]
```

### 17.5.2 Collection modification

---

If the collection is a modifiable (by the current user), then the **"addTo"** and **"removeFrom"** links will be provided.

If the collection is a Set (the common case, where entries cannot be duplicated), then the **"addTo"** link will be a PUT:

```
{
  "links": [ {
    "rel": ".../addTo;collection=\"items\"",
    "href": "http://~/objects/ORD/123/collections/items",
    "type": "application/json;profile=\".../object-collection\"",
    "method": "PUT",
    "arguments": {
      "value": null
    }
  },
  ...
],
...
}
```

If the collection is a List (the rarer case, where entries can be duplicated), then the **"addTo"** link will be a POST:

```
{
  "links": [ {
    "rel": ".../addTo;collection=\"items\"",
    "href": "http://~/objects/ORD/123/collections/items",
    "type": "application/json;profile=\".../object-collection\"",
    "method": "POST",
    "arguments": {
      "value": null
    }
  },
  ...
],
...
}
```

In both cases, the **"removeFrom"** link will be a DELETE:

```
{
  "links": [ {
    "rel": ".../removeFrom;collection=\"items\"",
    "href": "http://~/objects/ORD/123/collections/items",
    "type": "application/json;profile=\".../object-collection\"",
    "method": "DELETE",
    "arguments": {
      "value": null
    }
  },
  ...
],
...
}
```

# Restful Objects

To summarize:

JSON-Property	Description
<code>links[rel=.../add-to]</code>	link back to self to add to collection; not included if the collection is disabled
<code>links[rel=.../remove-from]</code>	link back to self to remove from collection; not included if the collection is disabled

If the collection is NOT modifiable (by the current user), then the representation will include a "**disabledReason**" json-property to indicate the reason (or just the literal "disabled") why the contents of the collection cannot be modified:

```
{
  ...
  "disabledReason":
    "Cannot add items to order that has already shipped",
  ...
}
```

where:

JSON-Property	Description
<code>disabledReason</code>	indicates the reason why the collection cannot be added to/removed from; only included if the collection is disabled

---

## 17.5.3 Domain model information

---

Domain model information is available through either the "**links**" or the "**extensions**" json-properties.

### 17.5.3.1 Simple scheme

Implementations that support the *simple* scheme provide extra data in the "**extensions**" json-properties. For example:

```
"extensions": {
  "friendlyName": "items",
  "description": "Line items (details) of the order",
  "returnType": "list",
  "elementType": "x.OrderItem",
  "pluralForm": "Order Items"
}
```

Note that the combination of the "**size**" json-property and the "**pluralForm**" json-property make it easy for a client to render useful summary information (e.g. "3 Customers").

See §A3.1.1 for the full definitions of these json-properties.



### 17.5.3.2 *Formal scheme*

Implementations that support the *formal* scheme §A3.1.2 provide an additional link only in the "**links**" json-property:

```
"links": [  
  {  
    "rel": "describedby",  
    "href": "http://~/domain-types/x.Order/collections/items",  
    "type": "application/json;profile=\"../type-collection\"",  
    "method": "GET"  
  },  
  ...  
]
```

which links to the domain collection description resource §D23.2 corresponding to this domain object collection.



## 18 ACTION RESOURCE & REPRESENTATION

The action resource can be used to obtain the detailed domain object action representation §18.2 for a particular domain object instance or for a particular domain service. This provides a *description* of the action only; to invoke it, the object action invoke sub-resource §C18.2 is used.

This resource is typically requested as a result of following a link from the domain object representation §14.4.

The endpoint URL for this resource for an action on a domain service is:

`/services/{serviceId}/actions/{actionId}`

where:

- `{serviceId}` is a unique identifier for the service
- `{actionId}` is the action identifier

The endpoint URL for this resource for an action on a domain object is:

`/objects/{domainType}/{instanceId}/actions/{actionId}`

where:

- `{domainType}` uniquely identifies the object's type, and
- `{instanceId}` uniquely identifies an object instance of that type
- `{actionId}` is the action identifier

### 18.1 HTTP GET

Obtain a detailed representation of an action §18.2.

---

#### 18.1.1 GET Request

---

##### 18.1.1.1 Query String

- **x-ro-domain-model** (optional, §A3.1)
  - "simple"
  - "formal"

##### 18.1.1.2 Headers

- Accept
  - application/json
  - application/json;profile=".../object-action"

##### 18.1.1.3 Body

- N/A

## 18.1.2 GET Success Response

As per §13.1 (200), returning an object action representation §18.2.

## 18.2 Representation

The domain object action representation provides full details of an action on a domain object instance, and provides links to resources that can invoke the action (if allowed).

The **Content-Type** for the representation is:

`application/json;profile=".../object-action"`

The links from the object action representation to other resources are as shown in the diagram below:

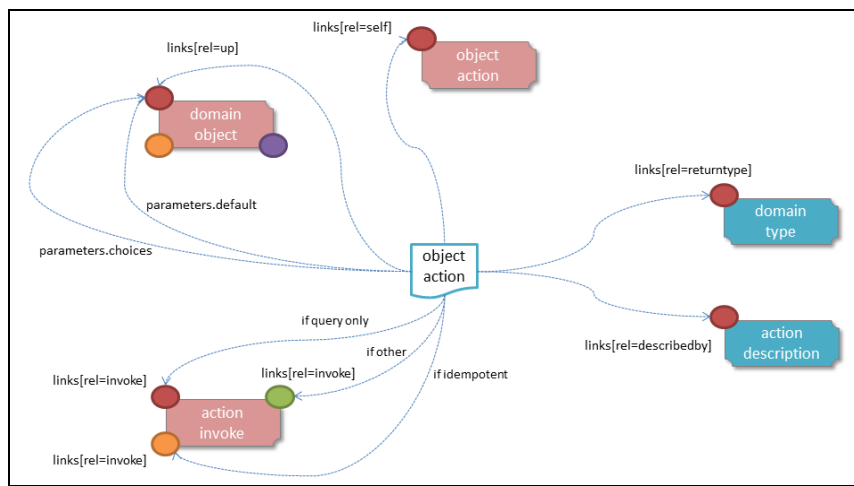


FIGURE 11: OBJECT ACTION REPRESENTATION

## Restful Objects

For example, the representation of Order's submit action might be:

```
{
  "id": "submit",
  ...
  "parameters": {
    ...
  },
  "disabledReason": ...,
  "links": [ {
    "rel": "self",
    "href": "http://~/objects/ORD/123/actions/submit",
    "type": "application/json;profile=\"../object-action\"",
    "method": "GET",
  }, {
    "rel": ".../invoke;action=\"submit\"",
    ...
  }, {
    "rel": "up",
    ...
  }
  ...
],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
<b>links</b>	list of links to other resources.
<b>links[rel=self]</b>	link to a resource that can generate this representation.
<b>id</b>	the action ID, to use when building templated URIs
<b>parameters</b>	map of parameters; discussed below § 18.2.1
<b>disabledReason</b>	(optional) if populated then indicates the reason why the action cannot be invoked.
<b>links[rel=.../invoke]</b>	(optional) is a link to invoke the action (if it is not disabled), § 18.2.2.
<b>links[rel=up]</b>	link to the object that is the owner of this action.
<b>extensions</b>	additional metadata about the resource

Both the "**links**" and the "**extensions**" json-properties may contain domain model information; this is discussed in § 18.2.3.

Restful Objects defines no standard child properties for the "**extensions**" json-property (other than any domain model information). Implementations are free to add further links/json-properties to both "**links**" and "**extensions**" as they require.

## 18.2.1 Action parameters

The action resource lists the parameter details in the "**parameters**" list:

```
"parameters": {
  "paramName1": {
    "choices": [ {
      "rel": ".../choice;action=\"action\";param=\"paramName1\"",
      ...
    }, {
      "rel": ".../choice;action=\"action\";param=\"paramName1\"",
      ...
    }, {
      "rel": ".../choice;action=\"action\";param=\"paramName1\"",
      ...
    }
  ],
  "default": {
    "rel": ".../default;action=\"action\";param=\"paramName1\"",
    ...
  },
  "links": [ ... ]
  "extensions": { ... }
},
"paramName2": {
  "choices": [ ... ],
  "default": { ... },
  "links": [ ... ]
  "extensions": { ... }
},
"paramName3": {
  "choices": [ ... ],
  "default": { ... },
  "links": [ ... ]
  "extensions": { ... }
}
]
```

where *paramName1*, *paramName2*, *paramName3* are the ids used as a unique key in the "**parameters**" map (also as used as the key in argument maps A2.9.2), with their value being the following map:

JSON-Property	Description
<b>choices</b>	an optional list of choices for the parameter argument
<b>default</b>	an optional value/link to act as the default for the parameter argument
<b>links</b>	list of links to other resources related to the action parameter
<b>extensions</b>	additional metadata about the action parameter

The rel values for choices and defaults specify both the *action* and parameter name in order to distinguish from other links.

The "**links**" and/or "**extensions**" json-property may hold domain model metadata; see §18.2.3.

### 18.2.2 Action invocation

---

If the action can be invoked then the **"rel" = "invoke"** link will contain a link by which the action can be invoked. This will be either a GET, a PUT or a POST dependent upon the action's semantics.

If the implementation can determine that the action is 'query-only', then a GET link should be provided:

```
{
  "links": [ {
    "rel": ".../invoke;action=\"recentOrder\"",
    "href": "http://~/objects/CUS/001/actions/recentOrder/invoke",
    "type": "application/json;profile=\".../action-result\"",
    "arguments": { ... },
    "method": "GET"
  },
  ...
],
...
}
```

If the implementation can determine that the action is idempotent then a PUT link will be provided:

```
{
  "links": [ {
    "rel": ".../invoke;action=\"makeRush\"",
    "href": "http://~/objects/ORD/123/actions/makeRush/invoke",
    "type": "application/json;profile=\".../action-result\"",
    "arguments": { ... },
    "method": "PUT"
  } ],
  ...
}
```

Finally, if the action to be invoked is neither query-only nor idempotent,(or if the implementation is unable to determine this), then a POST link will be provided:

```
{
  "links": [ {
    "rel": ".../invoke;action=\"submit\"",
    "href": "http://~/objects/ORD/123/actions/submit/invoke",
    "type": "application/json;profile=\".../action-result\"",
    "arguments": { ... },
    "method": "POST"
  } ],
  ...
}
```

## *"type" property*

The **"type"** json-property always indicates that the urn:org.restfulobjects:repr-types/action-result representation will be returned §19.4.

## *"arguments" property*

The **"arguments"** json-property has placeholders for the values of each of the arguments. Commonly, these values will be null - it is up to the client to determine the value to use when invoking the action. However the server may provide a default value.

To summarize:

JSON-Property	Description
link[.../rel=invoke]	link to invoke the action; not included if the action is disabled

If the action may NOT be invoked (for example because of the status of the object to which the action applied), then the representation should include a **"disabledReason"** json-property (or just the literal "disabled") why the action cannot be invoked:

```
{
  ...
  "disabledReason":
    "Cannot place order because customer has been blacklisted",
  ...
}
```

where:

JSON-Property	Description
disabledReason	indicates the reason why the action cannot be invoked; only included if the action is disabled.

---

## 18.2.3 Domain model information (for action)

---

Domain model information is available for both the action itself and also for each of the action parameters. In both cases the information is either under the **"links"** or under the **"extensions"** json-properties.



### 18.2.3.1 Simple scheme

Implementations that support the *simple* scheme provide extra data about the action in the **"extensions"** json-properties. For example:

```
"extensions": {
  "friendlyName": "Place order",
  "description": "Place a new order",
  "returnType": ...
  "elementType": ... // if returnType is 'list' or 'set'
  "pluralForm": ... // if returnType is 'list' or 'set'
  "hasParams": true,
  ...
}
```

In addition, such implementations may also provide extra data about each action *parameter* in that parameter's own **"extensions"** json-property. For example:

```
"parameters": {
  "product": {
    ...
    "extensions": {
      "friendlyName": "Product",
      "description": "The product being ordered",
      "returnType": ...
      "optional": false,
      "format": ... // for string params only
      "maxLength": ... // for string params only
      "pattern": ... // for string params only
    }
  }
  ...
}
```

See §A3.1.1 for the full definitions of these json-properties.

Implementations may also provide their own extensions.

### 18.2.3.2 Formal scheme

Implementations that support the *formal* scheme §A3.1.2 provide several additional links about the action in the **"links"** json-property. For example:

```
"links": [
  {
    "rel": ".../returntype",
    "href": "http://~/domain-types/x.OrderReceipt",
    "type": "application/json;profile=\".../domain-type\"",
    "method": "GET"
  },
  {
    "rel": "describedby",
    "href": "http://~/domain-types/x.Order/actions/submit",
    "type": "application/json;profile=\".../type-action\"",
    "method": "GET"
  },
  ...
]
```

## Restful Objects

---

In addition, implementations supporting the *formal* scheme may also provide extra data about each action *parameter* in that parameter's own **"links"** json-property.

For example:

```
"parameters": {
  "product": {
    ..
    "links": [
      {
        "rel": "describedby",
        "href":
          "http://~/domain-types/x.Order/actions/submit/params/product",
        "type":
          "application/json;profile=\"../action-param-description\"",
        "method": "GET"
      }
    ]
  }
  ...
}
```

## 19 ACTION INVOKE RESOURCE

The action invoke resource is used to either invoke an action on a particular domain object instance or a service, or to just validate arguments to an action without invoking it. It is usually obtained from the detailed representation §18.2 returned by an action resource §17.5.

The endpoint URL for this resource for a domain service is:

`/services/{serviceId}/actions/{actionId}/invoke`

where:

- `{serviceId}` is a unique identifier for the service
- `{actionId}` is the action identifier

The endpoint URL for this resource for a domain object is:

`/objects/{domainType}/{instanceId}/actions/{actionId}/invoke`

where:

- `{domainType}` uniquely identifies the object's type, and
- `{instanceId}` uniquely identifies an object instance of that type
- `{actionId}` is the action identifier

### 19.1 HTTP GET

Invoke an action and return a representation. Alternatively, validate the query arguments without invoking the action.

The action invoked must be query-only (does not modify any persisted objects); a typical example is to search for objects from a domain service repository.

The action cannot be void (it must return some representation).

---

#### 19.1.1 Request

---

The request can either be to invoke the action, or to request validation of arguments using the reserved **x-ro-validate-only** query parameter §A3.2.

##### 19.1.1.1 Query String

Query arguments should be formatted as a map (§A2.9.2), and encoded in the URL (§A2.9.2.5). Note that if any argument is a blob/clob, then its value must be in-lined (URL encoded for a blob)<sup>34</sup>.

---

<sup>34</sup> It seems highly unlikely that a query-only action would have a blob/clob argument, but it is theoretically allowable. One hi-tech use case could be to search images of Customers' faces against an image obtained from a webcam.

In addition, the following may optionally be included in the map:

- **x-ro-domain-model** (optional, §A3.1)
  - "simple"
  - "formal"
- **x-ro-validate-only** (optional, §A3.2)
  - "true"
    - the argument map can be incomplete; only those arguments provided will be validated.

### 19.1.1.2 Headers

- **Accept**
  - application/json
  - application/json;profile=".../action-result"

There is no need to pass **If-Match** for query-only actions.

### 19.1.1.3 Body

- N/A

---

## 19.1.2 Success Response

---

### 19.1.2.1 Status code

- 200 "OK"

### 19.1.2.2 Headers

- **Content-Length:**
  - size of the entity body
- **Content-Type** (if returning a domain object):
  - application/json;profile=".../action-result";x-ro-domain-type="yyy"
    - where yyy indicates the domain type (for object representations, §2.4.2);
      - the domain type id (if simple scheme)
      - URI of domain type (if formal scheme)

---

However, if the encoded size of the blob/clob exceeds the query string limit, then the action must be marked as idempotent in order that the argument be passed in the request body of a PUT.

- **Content-Type** (if returning a list of domain objects):
  - `application/json;profile=".../action-result";x-ro-element-type="yyy"`
    - where yyy indicates the domain type (for object representations, §2.4.2);
      - the domain type id (if simple scheme)
      - URI of domain type (if formal scheme)
- **Content-Type** (if returning a scalar or void):
  - `application/json;profile=".../action-result"`
- Caching headers:
  - TRANSACTIONAL, see §A2.13
    - if the object is transactional
  - NON\_EXPIRING, see §A2.13
    - if the implementation can determine that the returned representation is safe to cache (e.g. the returned objects are immutable reference data)

Note that an **ETag** is never returned for an action result. A client that wishes to modify the returned domain object must therefore re-retrieve it explicitly.

### 19.1.2.3 **Body**

As per §19.4.

## 19.2 HTTP PUT

Invoke an action and return a representation if the action returns a result. Alternatively, validate the query arguments but do not invoke the action.

The action invoked must be idempotent (though may have side-effects). An example might be `Order#submit()`, which (depending on how the application logic is written) might have the same post-conditions irrespective of whether the order has already been submitted or not.

---

### 19.2.1 Request

---

#### 19.2.1.1 **Query String**

- none

#### 19.2.1.2 **Headers**

- **Accept**
  - `application/json`
  - `application/json;profile=".../action-result"`
- **If-Match**
  - *timestamp digest*
    - obtained from **ETag** header of representation

## 19.2.1.3 **Body**

Arguments should be formatted as a map (§A2.9.2), and sent as the body (§A2.9.2.5). Note that if any argument is a blob/clob, then its value must be in-lined (URL encoded for a blob).

In addition:

- **x-ro-domain-model** (optional, §A3.1)
  - "simple"
  - "formal"
- **x-ro-validate-only** (optional, §A3.2)
  - "true"
    - only validate the request, do not invoke the action

---

## 19.2.2 **Success Response**

---

As per §19.1.2.

## 19.3 **HTTP POST**

Invoke an action, and return a representation if the action returns a result. Alternatively, validate the query arguments but do not invoke the action.

The action invoked can have side effects and need not be idempotent.

---

### 19.3.1 **Request**

---

#### 19.3.1.1 **Query String**

- none

#### 19.3.1.2 **Headers**

- **Accept**
  - application/json
  - application/json;profile=".../action-result"
- **If-Match**
  - *timestamp digest*
    - obtained from **ETag** header of representation

#### 19.3.1.3 **Body**

Arguments should be formatted as a map (§A2.9.2), and sent as the body (§A2.9.2.5). Note that if any argument is a blob/clob, then its value must be in-lined (URL encoded for a blob).

In addition:

- **x-ro-domain-model** (optional, §A3.1)
  - "simple"
  - "formal"

- **x-ro-validate-only** (optional, §A3.2)
  - "true"
    - only validate the request, do not invoke the action

---

### 19.3.2 Success Response

---

#### 19.3.2.1 Status code

Successfully invoking an action with possible side effects can return either a 200 or a 201.

- 200 "OK"
  - the action was successfully executed.
- 201 "Created"
  - only permitted when the action returns a domain object (that is "**resultType**" json-property is "**object**")
  - indicates that this object was newly created.

#### 19.3.2.2 Headers

- **Location:** (if returning 201)
  - URL of the newly-created action
- **Content-Length:**
  - size of the entity body
- **Content-Type** (if returning a domain object):
  - application/json;profile=".../action-result";x-ro-domain-type="yyy"
    - where yyy indicates the domain type (for object representations, §2.4.2);
      - the domain type id (if simple scheme)
      - URI of domain type (if formal scheme)
- **Content-Type** (if returning a list of domain objects):
  - application/json;profile=".../action-result";x-ro-element-type="yyy"
    - where yyy indicates the domain type (of the objects referenced in the list, §2.4.2);
      - the domain type id (if simple scheme)
      - URI of domain type (if formal scheme)
- **Content-Type** (if returning a scalar or void):
  - application/json;profile=".../action-result"
- Caching headers:
  - TRANSACTIONAL, see §A2.13
    - if the object is transactional
  - NON\_EXPIRING, see §A2.13
    - if the implementation can determine that the returned representation is safe to cache (e.g. the returned objects are immutable reference data)

Note that an **Etag** is never returned for an action result. A client that wishes to modify the returned domain object must therefore follow the self link on the in-lined object to retrieve that object directly as an object representation (which will then have an Etag).

### 19.3.2.3 *Body*

As per §19.4. If a 201 is returned, the "**resultType**" json-property must be "**object**".

## 19.4 Representation

If the "**x-ro-validate-only**" query parameter was passed in and the validation succeeded, then no representation will be returned. Instead:

- if the validation succeeded, then a 204 (success, no content) is returned
- If the validation failed then a representation will be returned, with a status code 400 (bad request).

See §13 for further details.

Otherwise (ie, if the invocation was not validate-only), then all action invocations will return an actionresult representation. This representation provides details of the action invocation, and (for non-void actions) also in-lines the representation of the result of the invocation.

For example:

```
{
  "links": [ {
    "rel": "self",
    "href":
"http://~/services/TaskRepository/actions/countUrgentTasksFor/invoke",
    "type": "application/json;profile=\"../action-result\"",
    "arguments": {
      "employee": {
        "href": "http://~/objects/EMP/090123"
      }
    }
  } ],
  "resultType": ...
  "value": ...,
  "extensions": { ... }
}
```

where:

JSON-Property	Description
links	list of links to other resources.



JSON-Property	Description
<b>links[rel=self]</b>	(optional) link to the action invocation resource that generated the representation (applies only to query-only actions)
<b>resultType</b>	either "object", "list", "scalar" or "void"
<b>result</b>	(optional) the action result itself. Not present if void action.
<b>extensions</b>	additional metadata about the representation.

The "**self**" link can be used as a bookmark so that the action can easily be resubmitted. However, the link is only included in the representation if the action is query-only. This is to prevent accidental bookmarking of links that if followed would result in side-effects.

The "**resultType**" indicates whether there is an in-lined representation (for an action returning a domain object, a list, a scalar) or none (if void).

Finally, the "**result**" holds the representation of the returned domain object, list, or scalar. This is discussed in sections below.

### 19.4.1 Action returning a Domain Object

If the action invocation returns a domain object, then the actionresult representation will in-line the domain object's representation (§14.1):

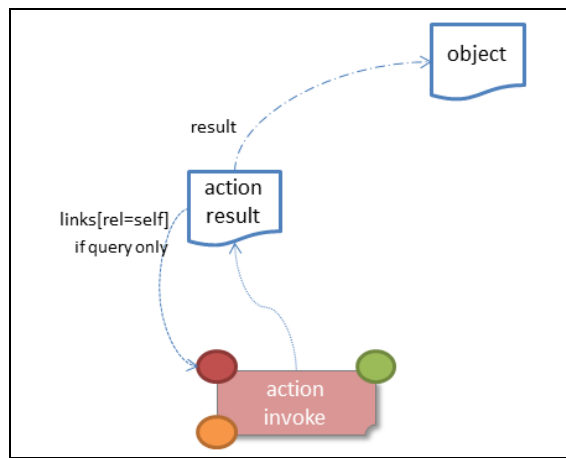


FIGURE 12: ACTION RESULT FOR OBJECT

For example, the following might be the result of invoking an action representing Customer's favoriteProduct() action:

```
{
  "links": [ {
    "rel": "self",
    "href": "http://~/objects/CUS/123/actions/favoriteProduct/invoke",
    "type": "application/json;profile=\"../action-result\"",
    "arguments": {},
    "method": "GET"
  }
],
  "resultType": "object",
  "result": {
    "links": [ {
      "rel": "self",
      "href": "http://~/objects/PRD/2468",
      "type": "application/json;profile=\"../object\"",
      "method": "GET"
    },
    ...
  ],
  "members": {
    ...
  },
  "extensions": { ... }
  ...
}
"extensions": { ... }
```

Note that this representation has two **self** links:

- links[rel=self]
  - is the link to the action invocation.
- result.links[rel=self]
  - is the link to the returned domain object.

If the action returned null, then the **result** json-property will still be present, but set to the JSON value null:

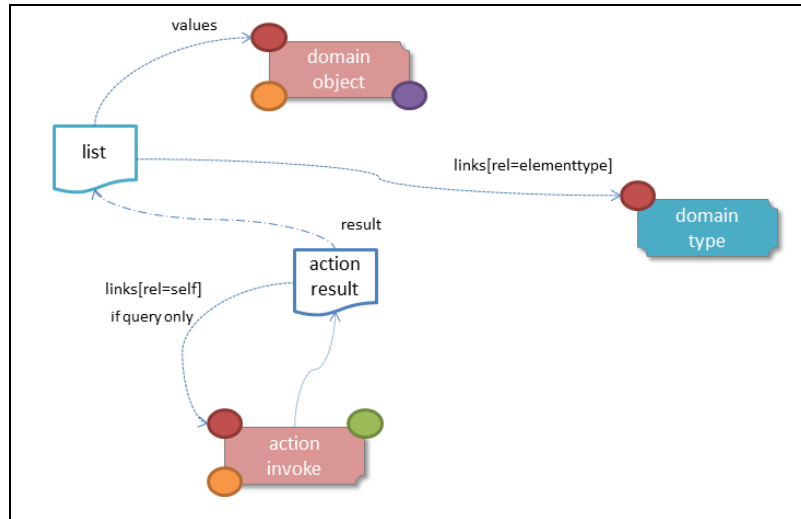
```
{
  ...
  "resultType": "object",
  "result": null
  ...
}
```

---

### 19.4.2 Action Returning a List

---

If the action invocation returns a list, then the actionresult representation will in-line a list representation (§B11):



**FIGURE 13: ACTION RESULT FOR LIST**

For example, the following might be the result of invoking an action resource § 17.5 representing CustomerRepository's findBlacklistedCustomers() action:

```
{
  "links": [ {
    "rel": "self",
    "href":
"http://~/services/CustomerRepository/actions/findBlackListedCustom
ers/invoke",
    "type": "application/json;profile=\"../action-result\"",
    "arguments": {},
    "method": "GET"
  }
],
  "resultType": "list",
  "result": {
    "links": [{
      "rel": "../element-type",
      "href": "http://~/domain-types/x.Customer",
      "type": "application/json;profile=\"../domain-type\"",
      "method": "GET"
    }
  ],
  "value": [ {
    "ref": "../element",
    "href": "http://~/objects/CUS/123",
    "type": "application/json;profile=\"../object\"",
    "method": "GET"
  }, {
    "ref": "../element",
    "href": "http://~/objects/CUS/456",
    "type": "application/json;profile=\"../object\"",
    "method": "GET"
  },
  ...
],
}
```

```
"extensions": { ... }  
},  
"extensions": { ... }  
}
```

Actions that return no links typically are expected to return an empty list:

```
{  
  ...  
  "resultType": "list",  
  "result": {  
    ...  
    "value": [ ]  
    ...  
  }  
  ...  
}
```

Although not recommended, it is also legal for actions to return a null list. In this case the **"result"** json-property will still be present, but will be set to the JSON value null:

```
{  
  ...  
  "resultType": "list",  
  "result": null  
  ...  
}
```

### 19.4.3 Action returning a Scalar Value

If the action invocation returns a scalar, then the actionresult representation will in-line a scalar representation (§B12):

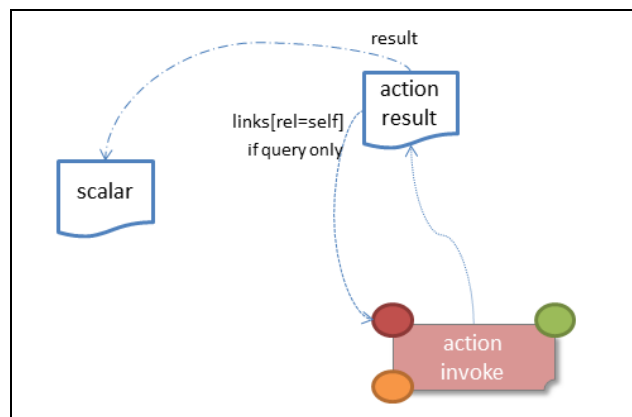


FIGURE 14: ACTION RESULT FOR SCALAR

## Restful Objects

---

For example, the TaskRepository's countUrgentTasksFor(Employee) action might generate the following representation:

```
{
  "links": [ {
    "rel": "self",
    "href":
"http://~/services/TaskRepository/actions/countUrgentTasksFor/invoke",
    "type": "application/json;profile=\"../action-result\"",
    "arguments": {
      "employee": {
        "href": "http://~/objects/EMP/090123"
      }
    },
    "method": "GET"
  } ],
  "resultType": "scalar",
  "result": {
    "links": [ {
      "rel": "../returntype",
      "href": "http://~/domain-types/int",
      "type": "application/json;profile=\"../domain-type\"",
      "method": "GET"
    } ],
    "value": 25,
    "extensions": { ... }
  },
  "extensions": { ... }
}
```

As for actions returning lists and domain objects, if the scalar return type is non-primitive and a null is returned, then the "result" json-property will be set to the JSON null value:

```
{
  ...
  "resultType": "scalar",
  "result": null
  ...
}
```

### 19.4.4 Action returning a Void

If the action invocation does not have a return type (known as a 'void' method in some programming languages), then the simple actionresult representation (with no in-lined representation) will be returned.

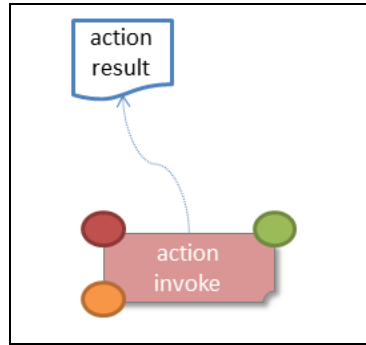


FIGURE 15: ACTION RESULT FOR VOID

For example, the Customer's `toggleBlacklistStatus()` action might generate the following representation:

```
{
  "links": [ {
    "rel": "self",
    "href":
      "http://~/objects/CUS/123/actions/toggleBlacklistStatus/invoke",
    "type": "application/json;profile=\"../action-result\"",
    "arguments": {}
  },
  ...
],
  "resultType": "void",
  "extensions": { ... }
}
```

Note that there is no **"result"** json-property.

# D

## DOMAIN TYPE RESOURCES





## 20 RESPONSE SCENARIOS

When a domain type resource is invoked, one of several responses can occur. These can be distinguished by the HTTP status return code

- 200 – Request succeeded, and generated a representation
- 404 – Type or type member not found or not visible
- 405 – Method not valid for the resource
- 406 – Client accepts only media types not generated by resource.

In addition, a 401 code may be returned for any resource if the user's credentials are invalid (that is, they have not authenticated themselves).

The following table indicates which of these status return codes may be generated for each resource:

	Method	Repr.	200	404	405	406
Domain Types	GET	§21	y	y	y	y
Domain Type	GET	§22	y	y	y	y
Type Property	GET	§23	y	y	y	y
Type Collection	GET	§24	y	y	y	y
Type Action	GET	§25	y	y	y	y
Type Action Parameter	GET	§26	y	y	y	y
Type Action Invoke	GET	§27	y	y	y	y

For a given status code, the specific headers and body returned by these resources vary little between the different resources; this is especially so for the failure scenarios (4xx and 5xx).

This section (§20) describes all the responses irrespective of resource called. Sections §21 to §27 identify the various request/response scenarios for each of the domain type resources. In each case they define the request URL, headers and body, and also identify the standard (success) response headers and body, if any.

### 20.1 Request succeeded, and generated a representation

For resources that return a body containing some representation.

#### 20.1.1 Status code

- 200 "OK"

---

## 20.1.2 Headers

---

- **Content-Length:**
  - size of the entity body
- **Content-Type:**
  - application/json;profile=".../xxx"
    - where xxx indicates the representation type, §2.4.1
- Caching headers:
  - NON\_EXPIRING, see §A2.13

---

## 20.1.3 Body (representation)

---

The representation will depend on the resource being requested.

## 20.2 Type or type member not found or not visible

This is the response if a requested type or type member does not exist, or if the object/member exists but is not visible based on the current user's credentials.

---

### 20.2.1 Status Code

---

- 404 "Not found"

---

### 20.2.2 Headers

---

- **Warning**
  - No such domain type {domainType}
  - No such property {propertyId}
  - No such collection {collectionId}
  - No such action {actionId}
  - No such action parameter {actionId, actionParamId}

---

### 20.2.3 Body

---

- empty

## 20.3 Resource has invalid semantics for method called

---

### 20.3.1 Status code

---

- 405 ("method not allowed")

---

### 20.3.2 Headers

---

- **Allow**
  - GET

---

### 20.3.3 Body

---

- empty

---

## 20.4 Not acceptable

The client has specified an **Accept** header that does not include a media type provided by the resource.

---

### 20.4.1 Status code

---

- 406 ("not acceptable")

---

### 20.4.2 Headers

---

- none

---

### 20.4.3 Body

---

- empty



## 21 DOMAIN TYPES RESOURCE

The domain types resource provides a list of all the domain types that are known to the system.

The endpoint URL for this resource is simply:

`/domain-types/`

### 21.1 HTTP GET

#### 21.1.1 Request

##### 21.1.1.1 Query String

- none

##### 21.1.1.2 Headers

- **Accept**
  - application/json
  - application/json;profile=".../type-list"

##### 21.1.1.3 Body

- N/A

#### 21.1.2 Successful Response

As per §20.1 (200); body as per §21.2.

### 21.2 Representation

The links from the typelist representation to other resources are as shown in the diagram below:

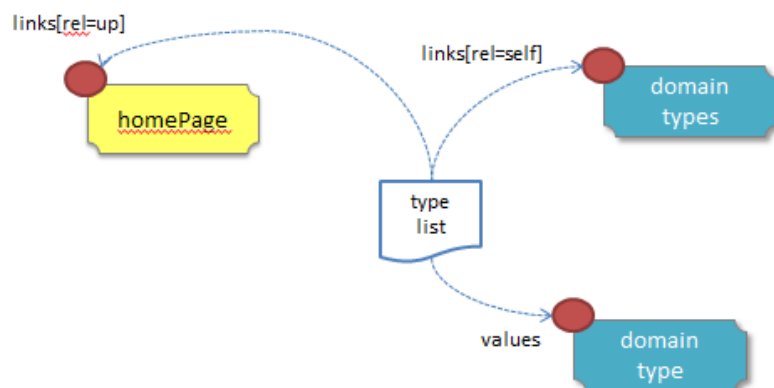


FIGURE 16: DOMAIN TYPE LIST REPRESENTATION

For example, the JSON representation of the list of domain types for a system looks something like:

```
{
  "links" : [ {
    "rel": "self",
    "href" : "http://localhost:8080/domain-types",
    "method" : "GET",
    "type" : "application/json;profile=\"../type-list\""
  }, {
    "rel": "up",
    "href": "http://~/",
    "type": "application/json;profile=\"../homepage\"",
    "method": "GET"
  }
],
  "value" : [ {
    "rel": "../domain-type",
    "href" : "http://~/domain-types/x.Customer",
    "method" : "GET",
    "type" : "application/json;profile=\"../domain-type\""
  }, {
    "rel": "../domain-type ",
    "href" : "http://~/domain-types/x.Order",
    "method" : "GET",
    "type" : "application/json; profile=\"../domain-type\""
  },
  ...
  "extensions" : { ... }
}
```

where:

JSON-Property	Description
links	list of links to resources
links[rel=self]	link to a resource that can obtain this representation
links[rel=up]	link to the homepage resource, B5.
values	List of links to domain types, §22
extensions	map of additional information about the resource.

### 21.3 Predefined Domain Types

There are a number of predefined "formal" domain type resources that correspond either to the built-in scalar types described §A2.5, or to lists and sets (for collections), or to void (for actions with void return type).

The following table shows the correlation in these cases:

Type	Simple scheme JSON datatype	Simple scheme 'format'	Formal scheme predefined type
string	String	string (the default)	http://~/domain-types/string
boolean	Boolean		http://~/domain-types/boolean

## Restful Objects

---

Type	Simple scheme JSON datatype	Simple scheme 'format'	Formal scheme predefined type
date-time	String	date-time	http://~/domain-types/date-time
date	String	date	http://~/domain-types/date
time	String	time	http://~/domain-types/time
epoch ms	String	utc-millisec	http://~/domain-types/utc-millisec
big integer	String	big-integer(n)	http://~/domain-types/big-integer(n)
big decimal	String	big-decimal(s,p)	http://~/domain-types/big-decimal(s,p)
blob	String	blob	http://~/domain-types/blob
clob	String	clob	http://~/domain-types/clob
decimal	Number	decimal	http://~/domain-types/decimal
int	Number	integer	http://~/domain-types/integer
list	---	---	http://~/domain-types/list
set	---	---	http://~/domain-types/set
void	---	---	http://~/domain-types/void

If the **"format"** json-property is omitted for a number, then the rules for interpreting that number as a float-point decimal or as an integer are as documented in the ECMAScript standard, §A2.5.

For large numbers, big-integer(n) specifies the scale n, while big-decimal(s,p) specifies the scale s and the precision p.

For example, big-integer(10) is numbers in the range 0 to 9,999,999,999, while big-decimal(10.2) is numbers in the range 0.00 to 99,999,999.99.

### *No representations are returned*

No representations are defined for any of the predefined domain type resources listed above; instead, a 204 (no content) will be returned. Clients are expected to have built-in support for these domain types (e.g. a calendar widget to render dates; a checkbox widget to render Booleans, and so on).





## 22 DOMAIN TYPE RESOURCE

The domain type resource represents the type of a domain object instance, within the metamodel. Its representation links through to other elements of the metamodel which represent the domain type's properties, collections and actions. These link back in turn to other domain type resources, for example representing the property type or an action's parameter types.

Clients can use the domain type to, for example, render a data element using a particular UI widget (datetime picker, textfield, spinner).

Server implementations are free to extend the representation as required, for example providing links to additional media (icons, videos and so forth).

The endpoint URL for this resource is:

`/domain-types/{domainType}`

where:

- `{domainType}` is either
  - the domain type id, or
  - is a built-in JSON type

### 22.1 HTTP GET

Obtain a representation of a domain type within the metamodel.

---

#### 22.1.1 Request

---

##### 22.1.1.1 *Query String*

- none

##### 22.1.1.2 *Headers*

- **Accept**
  - `application/json`
  - `application/json;profile=".../domain-type"`

##### 22.1.1.3 *Body*

- N/A

---

#### 22.1.2 Successful Response

---

As per §20.1 (200); body as per §22.2.

## 22.2 Representation

The links from the domain type representation to other resources are as shown in the diagram below:

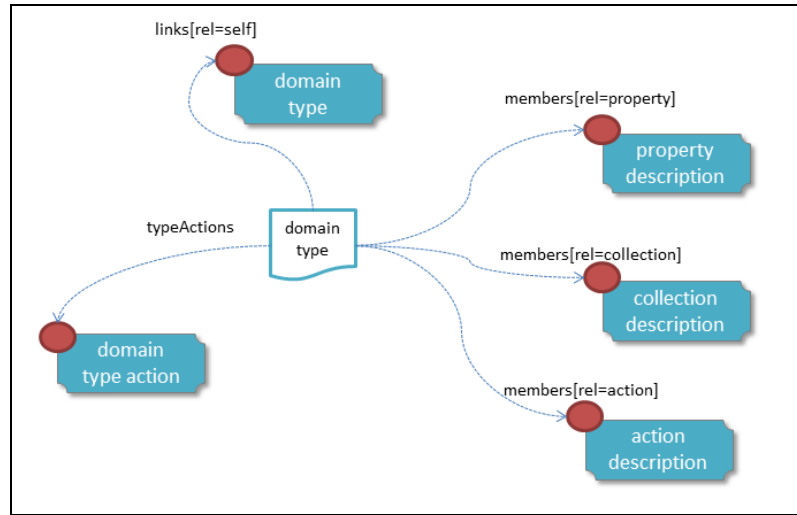


FIGURE 17: DOMAIN TYPE REPRESENTATION

For example, the JSON representation (for a Customer type) might look something like:

```
{
  "name": "x.Customer",
  "domainType": "CUS",
  "friendlyName": "Customer",
  "pluralName": "Customers",
  "description": "A customer with a registered account and confirmed billing details. ",
  "isService": false
  "members": {
    "firstName": {
      "rel": ".../property",
      "href": "http://~/domainType/x.Customer/properties/firstName",
      "type": "application/json;profile=\".../property-description\"",
      "method": "GET"
    },
    "recentOrders": {
      "rel": ".../collection",
      "href": "http://~/domainType/x.Customer/collections/recentOrders",
      "type": "application/json;profile=\".../collection-description\"",
      "method": "GET"
    }
  }
},
```

```
"blacklist": {
  "rel": ".../action",
  "href":
    "http://~/domainType/x.Customer/actions/blackList",
  "type":
    "application/json;profile=\".../action-description\"",
  "method": "GET"
},
...
},
"typeActions" : {
  "isSubtypeOf": {
    "rel": ".../invoke;typeaction=\"isSubtypeOf\"",
    "href":
      "http://~/domain-types/x.Customer/type-actions/isSubtypeOf/invoke",
    "method" : "GET",
    "type":
      "application/json;profile=\".../type-action-result\"",
    "arguments" : {
      "supertype" : {
        "href" : null
      }
    }
  },
  "isSupertypeOf": {
    "rel": ".../invoke;typeaction=\"isSupertypeOf\"", "href":
      "http://~/domain-types/x.Customer/type-actions/isSupertypeOf/invoke",
    "method" : "GET",
    "type" :
      "application/json;profile=\".../type-action-result\"",
    "arguments" : {
      "subtype" : {
        "href" : null
      }
    }
  }
},
...
},
"links": [ {
  "rel": "self",
  ...
}, {
  "rel": "icon",
  ...
},
...
],
"extensions": {
  ...
}
}
```

where:

JSON-Property	Description
links	list of links to resources
links[rel=self]	link to a resource that can obtain this representation
links[rel=icon]	(optional) link to an image representing a scalable icon for this type

# Restful Objects

JSON-Property	Description
<b>links[rel=help]</b>	(optional) link to a media resource providing help about the type
<b>name</b>	the fully qualified class name (or JSON type, if there is an equivalent)
<b>domainType</b>	the domainType string as used within templated URLs to access instances of this type (see resources in §C).
<b>friendlyName</b>	the singular form of the type, as would be suitable for rendering in a UI.
<b>pluralName</b>	the plural form of the type, as would be suitable for rendering in a UI.
<b>description</b>	a description of the type, e.g. to render as a tooltip.
<b>isService</b>	indicates whether the type is a domain service or not
<b>typeActions</b>	map of type action invocation resources, §27.
<b>members</b>	map of links to resources representing a description of a domain object property §D23.1, a domain object collection §D23.2, or a domain object action §D24.2.
<b>extensions</b>	map of additional information about the resource.

## *"links"*

The **"links"** list may contain links to a number of optional resources. For example:

```
"links": [ {
  "rel": "icon",
  "href": "http://~/images/Customer-32x32.jpg",
  "type": "image/jpg",
  "method": "GET"
}, {
  "rel": "help",
  "href": "http://~/videos/training/Customer-walkthru.mpg",
  "type": "audio/mpeg",
  "method": "GET"
},
...
]
```

Implementations are free to add their own resources to this list as they require.

## *"extensions"*

Restful Objects defines no standard json-properties for the **"extensions"** json-property, but implementations are free to add further links or extension json-properties to **"links"** and **"extensions"** as they require.

## 23 DOMAIN TYPE PROPERTY DESCRIPTION RESOURCE

The domain property description resource describes a property of a domain type within the metamodel.

Clients can use the domain property description's representation as hints when building a UI. For example, there will be links back to a domain type representing the property type; this can be used to select the relevant widget for that property. Or, the client can use information in the representation to apply client-side validation of declarative semantics (for example, mandatory properties, or regex patterns).

The endpoint URL for this resource is:

`/domain-types/{domainType}/properties/{propertyId}`

where:

- `{domainType}` is either
  - the domain type id, or
  - is a built-in JSON type
- `{propertyId}` identifies the property.

### 23.1 HTTP GET

Obtain a representation that describes a domain property within the metamodel.

---

#### 23.1.1 GET Request

##### 23.1.1.1 Query String

- none

##### 23.1.1.2 Headers

- **Accept**
  - `application/json`
  - `application/json;profile=".../property-description"`

##### 23.1.1.3 Body

- N/A

---

#### 23.1.2 Successful Response

As per §20.1 (200); body as per §23.2.

## 23.2 Representation

The links from the domain property description representation to other resources are as shown in the diagram below:

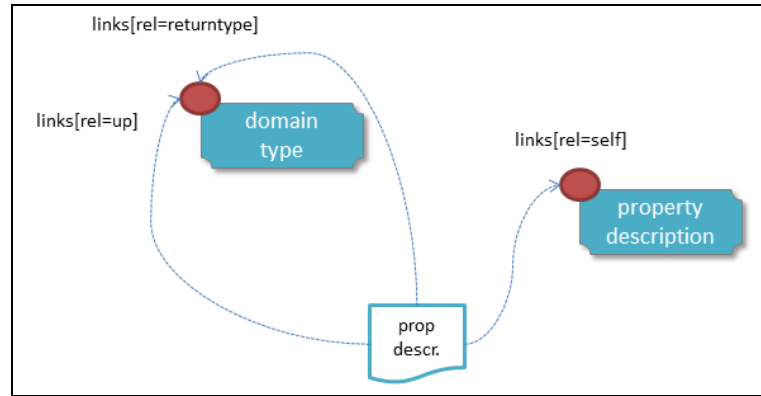


FIGURE 18: DOMAIN PROPERTY COLLECTION REPRESENTATION

The JSON representation (for the Order's deliveryTime property) looks something like:

```
{
  "id": "deliveryTime",
  "friendlyName": "Delivery Time",
  "description": "Time that the order will be delivered",
  "optional": false,
  "format": ... // for string properties only
  "maxLength": ... // for string properties only
  "pattern": ... // for string properties only
  "memberOrder": 1,
  "links": [ {
    "rel": "self",
    ...
  }, {
    "rel": "up",
    "href":
      "http://~/domain-types/x.Order",
    "type": "application/json;profile=\".../domain-type\"",
    "method": "GET"
  }, {
    "rel": ".../returntype",
    "href":
      "http://~/domain-types/string",
    "type": "application/json;profile=\".../domain-type\"",
    "method": "GET"
  },
  ...
],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
links	list of links to resources

## Restful Objects

---

JSON-Property	Description
links[rel=self]	link to a resource that can obtain this representation
id	the Id of this property
friendlyName	the property name, formatted for rendering in a UI.
description	a description of the property, e.g. to render as a tooltip.
optional	indicates whether the property is optional
maxLength	for string properties, indicates the maximum allowable length. A value of 0 means unlimited.
pattern	for string properties, indicates a regular expression for the property to match.
memberOrder	a presentation hint recommending the relative order to display each member (clients are not obliged to follow this).
format	for properties returning a string or number value, indicates how to interpret that value §A2.5.
links[rel=up]	link to the domain type which owns this property
links[rel=.../return-type]	link to the domain type of which this property holds a value (ie, its return type)
links[rel=help]	(optional) link to a media resource providing help about the property
extensions	additional information about the resource.

### *"extensions"*

Restful Objects defines no standard json-properties within "**extensions**", but implementations are free to add further links/ json-properties to "**links**" and "**extensions**" as they require.

One possible example is to specify which properties should appear as table columns when the domain type in question is the element type of a collection or list being rendered as such.





## 24 DOMAIN TYPE COLLECTION DESCRIPTION RESOURCE

The domain collection description resource represents a description of a domain collection, within the metamodel.

Clients can use the domain collection description's representation as hints when building a UI. For example, there will be links back to the domain type representing the collection's element type; this can be used by a client to determine columns for a table view. Or, the client can use information in the representation in order to apply client-side validation of declarative semantics (for example, minimum or maximum cardinality of a collection).

The endpoint URL for this resource is:

`/domain-types/{domainType}/collections/{collectionId}`

where:

- `{domainType}` is either
  - the domain type id, or
  - is a built-in JSON type
- `{collectionId}` identifies the collection.

### 24.1 HTTP GET

Obtain a representation of a description of a domain collection, within the metamodel.

---

#### 24.1.1 GET Request

---

##### 24.1.1.1 Query String

- none

##### 24.1.1.2 Headers

- **Accept**
  - `application/json`
  - `application/json;profile=... /collection-description`

##### 24.1.1.3 Body

- N/A

---

#### 24.1.2 Successful Response

---

As per §20.1 (200); body as per §24.2.

## 24.2 Representation

The links from the domain collection description representation to other resources are as shown in the diagram below:

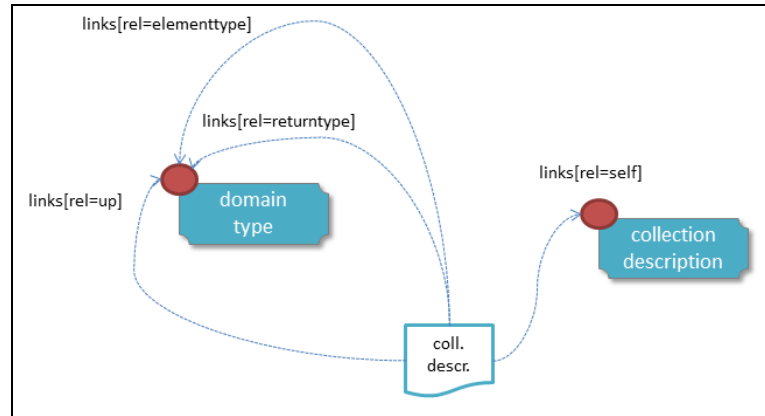


FIGURE 19: DOMAIN COLLECTION DESCRIPTION REPRESENTATION

The JSON returned representation (for the Order's items collection) might look something like:

```
{
  "id": "items",
  "friendlyName": "items",
  "plural form": "Order items",
  "description": "Line items (details) of the order",
  "memberOrder": 3,
  "links": [ {
    "rel": "self",
    ...
  }, {
    "rel": "up",
    "href": "http://~/domain-types/x.Order",
    "type": "application/json;profile=\"../domain-type\"",
    "method": "GET"
  },
  {
    "rel": "../returntype",
    "href": "http://~/domain-types/list",
    "type": "application/json;profile=\"../domain-type\"",
    "method": "GET"
  },
  {
    "rel": "../elementtype",
    "href": "http://~/domain-types/x.OrderItem",
    "type": "application/json;profile=\"../domain-type\"",
    "method": "GET"
  },
  {
    "rel": "help",
    "href": "http://~/videos/training/order-items-explained.mpg",
    "type": "audio/mpeg",
    "method": "GET"
  },
  ...
],
}
```

# Restful Objects

```
"extensions": { ... }  
}
```

where:

JSON-Property	Description
<b>links</b>	list of links to other resources.
<b>links[rel=self]</b>	link to a resource that can obtain this representation
<b>id</b>	the Id of this collection
<b>friendlyName</b>	the collection name, formatted for rendering in a UI.
<b>pluralForm</b>	the pluralized form of the element type within the collection/list.
<b>description</b>	a description of the collection, e.g. to render as a tooltip.
<b>memberOrder</b>	a presentation hint as to the relative order to display each member
<b>links[rel=up]</b>	link to the domain type which owns this property
<b>links[rel=.../return-type]</b>	link to the domain type for list or for set.
<b>links[rel=.../element-type]</b>	link to the domain type of the objects contained in the collection
<b>links[rel=help]</b>	(optional) link to a media resource providing help about the property
<b>extensions</b>	additional information about the resource.

## *"extensions"*

Restful Objects defines no standard json-properties within **"extensions"**, but implementations are free to add further links/properties to **"links"** and **"extensions"** as they require.



## 25 DOMAIN TYPE ACTION DESCRIPTION RESOURCE

The domain action description resource represents an action of a domain type, within the metamodel.

Clients can use the domain action description's representation as hints when building a UI. For example, the representation of an object action may include a link to that action's return type.

The endpoint URL for this resource is:

`/domain-types/{domainType}/actions/{actionId}`

where:

- `{domainType}` is either
  - the domain type id, or
  - is a built-in JSON type
- `{actionId}` identifies the action.

### 25.1 HTTP GET

Obtain a representation of a description of a domain action, within the metamodel.

---

#### 25.1.1 GET Request

##### 25.1.1.1 *Query String*

- none

##### 25.1.1.2 *Headers*

- **Accept**
  - `application/json`
  - `application/json;profile=".../action-description"`

##### 25.1.1.3 *Body*

- N/A

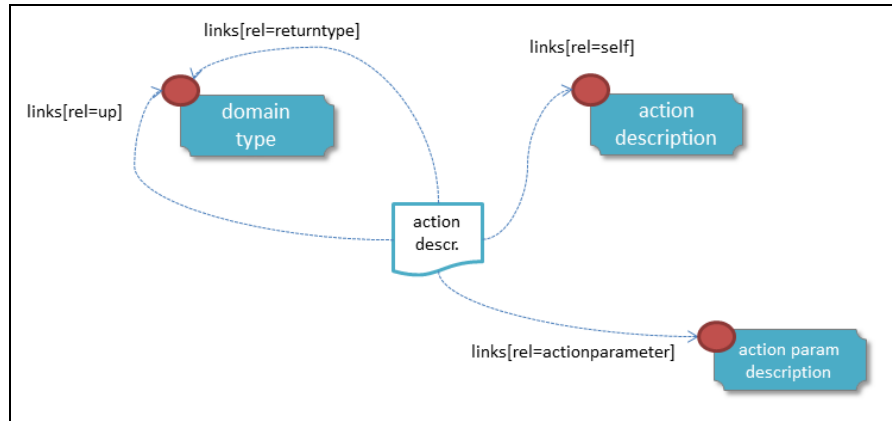
---

#### 25.1.2 Successful Response

As per §20.1 (200); body as per §25.2.

## 25.2 Representation

The links from the domain action description representation to other resources are as shown in the diagram below:



**FIGURE 20: DOMAIN ACTION DESCRIPTION REPRESENTATION**

The JSON representation of the domain action description (for the Order's submit action) might look something like:

```
{
  "id": "submit",
  "friendlyName": "Submit",
  "description": "...",
  "hasParams": true,
  "memberOrder": 5,
  "links": [ {
    "rel": "self",
    ...
  }, {
    "rel": "up",
    "href": "http://~/domain-types/x.Order",
    "type": "application/json;profile=\".../domain-type\"",
    "method": "GET"
  },
  {
    "rel": ".../return-type",
    "href": "http://~/domain-types/x.OrderReceipt",
    "type": "application/json;profile=\".../domain-type\"",
    "method": "GET"
  },
  {
    "rel": "help",
    "href": "http://~/videos/training/Order-submit.mpg",
    "type": "audio/mpeg",
    "method": "GET"
  },
  ...
],
  "parameters": {
```

# Restful Objects

```
{
  "ship": {
    "rel": ".../action-param;param=\"ship\"",
    "href": "http://~/domain-types/x.Order/submit/params/ship",
    "type":
      "application/json;profile=\".../action-param-description\"",
    "method": "GET"
  },
  "rush": {
    "rel": ".../action-param;param=\"rush\"",
    "href": "http://~/domain-types/x.Order/submit/params/rush",
    "type":
      "application/json;profile=\".../action-param-description\"",
    "method": "GET"
  },
  ...
},
"extensions": { ... }
}
```

where:

JSON-Property	Description
links	list of links to resources.
links[rel=self]	link to a resource that can obtain this representation
id	the Id of this action
friendlyName	the action name, formatted for rendering in a UI.
pluralForm	(optional) for actions returning collections the pluralized form of the element type within the collection/list.
description	a description of the action, e.g. to render as a tooltip.
hasParams	whether the action has parameters
memberOrder	a presentation hint as to the relative order to display each member
links[rel=up]	link to the domain type which owns this action
links[rel=.../return-type]	link to the action's return type
links[rel=.../element-type]	(optional) link to the element type if the action returns a collection.
links[rel=help]	(optional) link to a media resource providing help about the action
parameters	map of links to parameter details §26
extensions	map of additional information about the resource.

## "extensions"

Restful Objects defines the no standard json-properties within "**extensions**", but implementations are free to add further links/json-properties to "**links**" and "**extensions**" as they require.





## 26 DOMAIN TYPE ACTION PARAMETER DESCRIPTION RESOURCE

The domain action parameter description resource represents a description of a parameter of an action on a domain type, within the metamodel.

Clients can use the domain action parameter description representations as hints when building a UI. For example, there will be links back to the domain type representing each of the action parameter return types; this might be used in order to select the appropriate widgets when building a UI. Or, the client can use information in the representation to apply client-side validation of declarative semantics (for example, mandatory parameters, or regex patterns).

The endpoint URL for this resource is:

`/domain-types/{domainType}/actions/{actionId}/params/{paramName}`

where:

- `{domainType}` is either
  - the domain type id, or
  - is a built-in JSON type
- `{actionId}` identifies the action
- `{paramName}` is the named action parameter

### 26.1 HTTP GET

Obtain a representation of a description of a domain action parameter, within the metamodel.

---

#### 26.1.1 GET Request

---

##### 26.1.1.1 Query String

- none

##### 26.1.1.2 Headers

- **Accept**
  - `application/json`
  - `application/json;profile=".../action-param-description"`

##### 26.1.1.3 Body

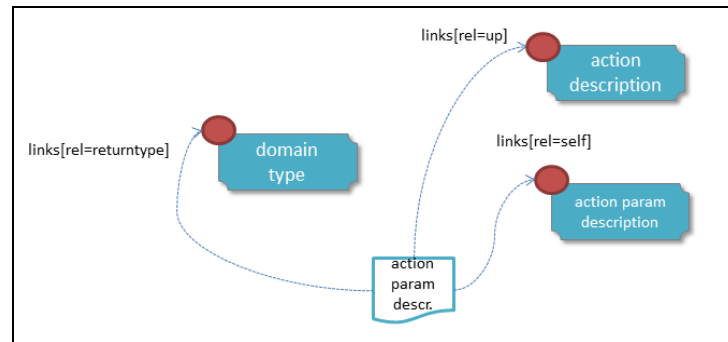
- N/A

## 26.1.2 Successful Response

As per §20.1 (200); body as per §26.2.

## 26.2 Representation

The links from the domain action parameter description representation to other resources are as shown in the diagram below:



**FIGURE 21: DOMAIN ACTION PARAMETER DESCRIPTION REPRESENTATION**

The JSON representation returned (for example, if the Order's submit action's first parameter is a "shipMethod" string parameter) might look something like:

```
{
  "id": "submit-shipMethod",
  "number": 0,
  "name": "shipMethod",
  "friendlyName": "Ship Method",
  "description": "...",
  "optional": false,
  "format": ...           // for string params only
  "maxLength": ...        // for string params only
  "pattern": ...          // for string params only
  "links": [ {
    "rel": "self",
    ...
  }, {
    "rel": "up",
    "href": "http://~/domain-types/x.Order/actions/submit",
    "type": "application/json;profile=\".../action-description\"",
    "method": "GET"
  }, {
    "rel": ".../returntype",
    "href": "http://~/domain-types/string",
    "type": "application/json;profile=\".../domain-type\"",
    "method": "GET"
  }, {
    "rel": "help",
    "href":
      "http://~/videos/training/order-submit-shipMethod.mpg ",
    "type": "audio/mpeg",
    "method": "GET"
  },
  ...
],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
links	list of links to other resources.
links[rel=self]	link to a resource that can obtain this representation
id	the Id of this action parameter (typically a concatenation of the parent action Id with the parameter name)
name	the name of the parameter
number	the number of the parameter (starting from 0)
friendlyName	the action parameter name, formatted for rendering in a UI.
description	a description of the action parameter, e.g. to render as a tooltip.
optional	indicates whether the action parameter is optional
maxLength	for string action parameters, indicates the maximum allowable length. A value of 0 means unlimited.
pattern	for string action parameters, indicates a regular expression for the property to match.

## Restful Objects

---

JSON-Property	Description
<b>format</b>	for action parameters requiring a string or number value, indicates how to interpret that value §A2.5.
<b>links[rel=up]</b>	link to the action that owns this parameter
<b>links[rel=.../return-type]</b>	link to the action parameter's return type
<b>links[rel=help]</b>	(optional) link to a media resource providing help about the action parameter
<b>extensions</b>	map of additional information about the resource.

### *"extensions"*

Restful Objects defines the no standard json-properties within "**extensions**", but implementations are free to add further links or json-properties as they require.

## 27 DOMAIN TYPE ACTION INVOKE RESOURCE

The domain type action invoke resource represents an action that can be invoked on the domain type itself. Conceptually it is similar to the domain object action invoke resource, §C19, but the action is on the type rather than an instance of the type.

Restful Objects defines the following type actions.

Type action	See	Description
<b>isSubtypeOf()</b>	§27.1	to determine if a domain type is a subtype of (or the same as) a given type.
<b>isSupertypeOf()</b>	§27.2	to determine if a domain type is a supertype of (or the same as) a given type.

The endpoint URL for this resource is:

`/domain-types/{domainType}/type-actions/{typeactionId}/invoke`

where:

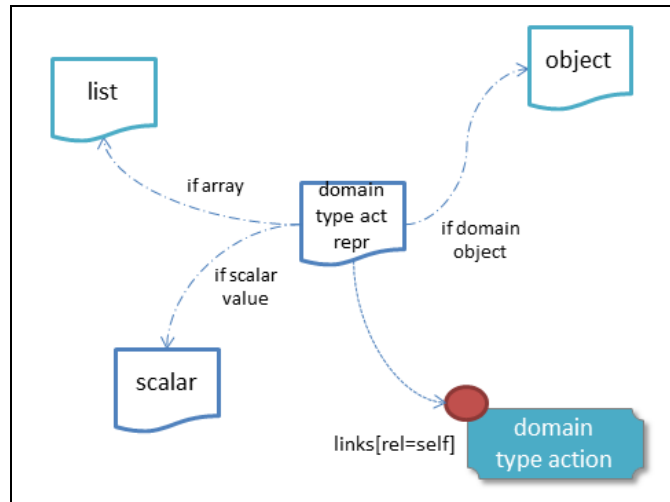
- `{domainType}` is either
  - the fully qualified type name, or
  - is a built-in JSON type
- `{typeactionId}` identifies the action

Both of the defined type actions take a single argument representing a link to the supertype.

## Restful Objects

As for the invocation of object actions (§C19), type actions return a representation that links back to the type action and also to the result of the action. Both the currently defined type actions return a scalar §B12.

The links from the domain type action result representation to other resources are as shown in the diagram below:



**FIGURE 22: DOMAIN TYPE ACTION REPRESENTATION**

The representations returned by type actions follow the format:

```
{
  "links" : [ {
    "rel" : "self",
    "href" : "http://~/domain-types/x.Customer/typeactions/.../invoke",
    "method" : "GET",
    "type": "application/json;profile=\"../type-action-result\"",
    "arguments" : ...
  } ],
  "id": ...,
  "value" : ...,
  "extensions" : { ... }
}
```

where:

JSON-Property	Description
links	list of links to other resources.
links[rel=self]	link to a resource that can obtain this representation
id	the typeActionId of this action
value	the result of the action
extensions	map of additional information about the resource.

## 27.1 HTTP GET isSubtypeOf()

Obtain a representation resulting from the invocation of the isSubtypeOf() domain type action.

---

### 27.1.1 GET Request

#### 27.1.1.1 URL and Query String

The endpoint URL for this resource is:

`/domain-types/{domainType}/type-actions/isSubtypeOf/invoke`

This resource requires a parameter "supertype" which can be specified either in the simple style, or in the formal style.

##### *Simple Style*

If query arguments are specified using the simple style (§A2.9.1) then the query string is simply:

`?supertype=xxx`

where xxx is the domain type id of the supertype.

##### *Formal Style*

If query arguments are specified using the formal style (§A2.9.2) then the supertype should be specified as a map and then the map encoded as a URL (§A2.9.2.5).

For example:

```
{
  "supertype": {
    "value": {
      "href": "http://~/domain-types/x.BasketOwner",
    }
  }
}
```

Note that the value should be a link to the supertype, rather than simply the name of the supertype.

#### 27.1.1.2 Headers

- **Accept**
  - application/json
  - application/json;profile=".../type-action-result"

#### 27.1.1.3 Body

- N/A

## 27.1.2 GET Response

### 27.1.2.1 Status Code

- 200 "OK"

### 27.1.2.2 Headers

- **Content-Type**
  - application/json;profile=".../type-action-result"
- Caching headers:
  - NON\_EXPIRING, see §A2.13
    - the structure of a domain type will not vary between deployments

### 27.1.2.3 Body (representation)

The JSON representation returned The JSON representation returned is a type-action-result representation (as described earlier) with a scalar representation §B12 in-lined.

For example, if checking that the Customers domain type is a subtype of BasketOwner interface type, then the returned representation might look something like:

```
{
  "links" : [ {
    "rel" : "self",
    "href" : "http://~/domain-types/x.Customer
              /typeactions/isSubtypeOf/invoke",
    "method" : "GET",
    "type" : "application/json;profile=\".../type-action-result\"",
    "arguments" : {
      "supertype" : {
        "href" : "http://~/domain-types/x.BasketOwner"
      }
    }
  } ],
  "id": "isSubtypeOf",
  "value" : true,
  "extensions" : { ... }
}
```

where:

JSON-Property	Description
id	the literal "isSubtypeOf" for this type action
value	a scalar boolean value.

and other properties are as described earlier.



---

## 27.1.3 GET Not found Response

---

### 27.1.3.1 Status Code

- 404 "Not found"

### 27.1.3.2 Headers

- **Warning:**
  - No such domain type {domainType}.
  - No such domain type action {actionId} in domain type {domainType}
  - No such super/subtype

### 27.1.3.3 Body

empty

## 27.2 HTTP GET isSupertypeOf()

Obtain a representation resulting from the invocation of the isSupertypeOf() domain type action.

---

## 27.2.1 GET Request

---

### 27.2.1.1 URL and Query String

The endpoint URL for this resource is:

`/domain-types/{domainType}/type-actions/isSupertypeOf/invoke`

This resource requires a parameter "subtype" which can be specified either in the simple style, or in the formal style.

#### *Simple Style*

If query arguments are specified using the simple style (§A2.9.1) then the query string is simply:

`?subtype=xxx`

where xxx is the domain type id of the subtype.

#### *Formal Style*

If query arguments are specified using the formal style (§A2.9.2) then the subtype should be specified as a map and then the map encoded as a URL (§A2.9.2.5).

For example:

```
{
  "subtype": {
    "value": {
      "href": "http://~/domain-types/x.Customer",
    }
  }
}
```

Note that the value should be a link to the subtype, rather than simply the name of the subtype.

### 27.2.1.2 Headers

- **Accept**
  - application/json
  - application/json;profile=".../type-action-result"

### 27.2.1.3 Body

- N/A

---

## 27.2.2 GET Response

---

### 27.2.2.1 Status Code

- 200 "OK"

### 27.2.2.2 Headers

- **Content-Type**
  - application/json;profile=".../type-action-result"
- Caching headers:
  - NON\_EXPIRING, see §A2.13
    - the structure of a domain type will not vary between deployments

### 27.2.2.3 Body (representation)

The JSON representation returned The JSON representation returned is a typeactionresult representation (as described earlier) with a scalar representation §B12 in-lined.

For example, if checking that the BasketOwner domain type is a supertype of the Customer domain type, then the returned representation might look something like:

```
{
  "links" : [ {
    "rel" : "self",
    "href" : "http://~/domain-types/x.BasketOwner
              /typeactions/isSupertypeOf/invoke",
    "method" : "GET",
    "type" : "application/json;profile=\"../type-action-result\"",
    "arguments" : {
      "supertype" : {
        "href" : "http://~/domain-types/x.Customer"
      }
    }
  } ],
  "id": "isSupertypeOf",
  "value" : true,
  "extensions" : { ... }
}
```

where:

JSON-Property	Description
id	the literal "isSupertypeOf" for this type action
value	a scalar boolean value.

and other properties are as described earlier.

---

### 27.2.3 GET Not found Response

---

#### 27.2.3.1 Status Code

- 404 "Not found"

#### 27.2.3.2 Headers

- **Warning:**
  - No such domain type {domainType}.
  - No such domain type action { actionId} in domain type {domainType}
  - No such super/subtype

#### 27.2.3.3 Body

empty



# E

## DISCUSSIONS



# 28 HATEOAS vs WEB APIs

*Comparing two styles of distributed system.*

## 28.1 HATEOAS (Hypermedia Controls)

REST-based systems are typically designed to have a single or small number of starting (home page) resources, from which all other resources can be traversed. This is REST's HATEOAS concept: *hypertext as the engine of application state*. When you use a web browser you can click through links to get to the next page; HATEOAS says that a REST client should be able to do likewise to follow a link to another resource.

Said more generally, links are an example of a hypermedia control by which the REST client can navigate the resources. The client knows what the link represents through its "**rel**" (relation) property; however the value of the URL is opaque.

This style of REST is supported by Restful Objects in that all resources in Restful Objects are discoverable from the Home Page resource §B5, acting as a well-known starting point. The rels are defined in §A2.7.1.

All other resources can be obtained from the home page, for example by following the links that represent reference properties and collections between domain objects, or by invoking domain object or service actions that can return references to other objects. Links are only included in representations if it makes sense for the client to follow them.

## 28.2 Web APIs (Templated URIs)

Some practitioners take the view that a pure-HATEOAS design places too much responsibility on the part of the client (having to traverse through multiple sets of representations from the start page), and that this can result in slower performance (for much the same reasons).

This isn't strictly true; a client is free to cache the value of a link between interactions (rather than find it out "from first principles" each time). Said another way: clients may bookmark links in order to move from one representation to the next (though they should be prepared to recover if the link is no longer valid).

Even so, many prefer a system that defines a set *templated URIs* in the form:

`path/{pathParam1}/pathPart/{pathParam2}/...`

So long as the client has arguments for each of the {pathParam}'s then it can directly construct the URL and invoke it.

## Restful Objects

---

It is highly debatable whether this style of system should be called REST; a better name is probably an HTTP-based web API. Nevertheless it is popular, and it is practicable. It is also supported by Restful Objects; the URLs for each of the resources included in the specification are well-defined, and the information needed to construct the URLs (such as the object identifiers) are provided as properties in the JSON representations.

Implementations of the spec may also choose to provide additional support - for example leveraging the capabilities of lower-level frameworks such as JBoss RESTEasy (<http://www.jboss.org/resteasy>) that provide explicit support for client-side templated URIs.



# 29 PERSONAL VS SHARED STATE

*Different types of resource state, what is not resource state, and discussion of the "domain-model rest anti-pattern" paper.*

## 29.1 Resources representing Shared State

In Roy Fielding's thesis that originally describes the REST architectural style, the definition of a resource is very general, with the focus being that resources may be identified and thus may be linked to.

The examples of resources given by Fielding in his thesis tend towards shared information, for example a document, an image, "today's weather in Los Angeles", or "revision 1.2.7 of a source code file".

These examples of resources are shared in the sense that any user could ask the system for a representation of the resource, and would obtain broadly the same information. Put another way, these resources correspond to domain entities, with the representations being a projection of the state of those entities. Their representation may need to be versioned to allow clients to evolve independently of server, but that is a separate issue (see §30).

Most of the examples in this specification are of domain entities: customer, order, product and so forth. The representations of these entities have hypermedia controls (links) to enable a client to navigate between entities, either as a result of following a property or collection link, or as the result of invoking an action. The links available may vary depending upon both the state of the resource, and upon the authorization of the user requesting the representation.

For example, the client can use the link representing the Order's `placedBy` property to find the Customer that placed the order, or can use the link representing the Order's `items` collection to find the items within the Order. However, an Order that has not yet been placed might suppress the Customer link.

Links not only represent static relationships between entities, they can also represent entity behaviour. So, the Order could have been created in the first place by the client following the link representing the Customer's `placeOrder()` action. However, a user with insufficient privileges might not see the `placeOrder()` link.

In this way, resources representing domain entities fully support the HATEOAS constraint (§28.1) of RESTful systems.

### 29.2 Resources representing Personal State

Whereas domain entities constitute shared state (available to any requesting user), some state is private to a given user and should not be accessible to other users.

The classic example is that of a ShoppingCart. Each user may have access to a resource representing "their" ShoppingCart, but should not be able to access (or even determine the existence of) other users' ShoppingCarts.

Resources that capture such "personal" state often double up as a means to take the user through the process of completing a user goal. For example, the initial representation of a ShoppingCart may provide links to browse for products to add to the cart. Once some products have been added, it may additionally provide a link to checkout the cart. Once the checkout is started, the links will change to the various steps of the checkout process (payment, delivery options and so on).

Restful Objects does not specify how implementations should protect personal state, but there are at least two possible approaches.

In the first, implementations can exploit the fact that URLs are opaque, and encrypt the instance identifier component. This is discussed further and in more general terms in §30.

The second approach is for the implementation to provide a mechanism to distinguish between a resource that holds personal state and one that holds shared state, and ensured that a user is only ever served up "their" personal resources. This would be analogous to querying from a database view that restricts rows by `userid`:

```
create view MyPersonalResource
as
select *
  from AllPersonalResources
 where user_id = @@user_id      -- current user id
```

One way to implement this could be through a reserved/annotated "UserId" property (or method) in the domain object; for example:

```
public class ShoppingCart {
    @UserId
    public int getPersonalTo() { ... }

    public List<Item> getItems() { ... }
    ...
}
```

If the `@UserId` property is present then the implementation infers that the object is personal to that user, and never returns it as a resource if requested by any other user.

### 29.3 Application State

Personal state, discussed above, is not the same as application state, though the distinction is subtle. In a blog post from 2008<sup>35</sup>, Roy Fielding wrote:

*Don't confuse application state (the state of the user's application of computing to a given task) with resource state (the state of the world as exposed by a given service). They are not the same thing.*

At first glance one might consider that this definition does not allow resources to represent personal state; after all, a personal state resource exists to manage the state of a user's application. However, we should not confuse the state of a resource on the server with the state of the client as a result of consuming that representation. Put another way: if a user accesses their shopping cart with a web browser, then the application state is not the shopping cart resource, it is the in-memory DOM structure within their browser.

What this also means is that the phrase "state of the world" in Fielding's definition is quite narrow: it should be taken to mean "as observed by a given user" rather than "as observed by any user". REST does therefore allow for resources to have either personal state or to have shared state.

### 29.4 Domain Model Resources an Anti-Pattern?

Some REST practitioners argue<sup>36</sup> that exposing domain model objects through REST is an anti-pattern. Or to use the terminology introduced in this chapter, the argument is that resources should only expose personal state, never shared state. To this, we strongly disagree.

For a system to be called RESTful it must obey the HATEOAS constraint and provide hypermedia controls to enable the client to navigate its resources. As described above, both personal state resources (shopping carts) and shared state resources (customer, order, product) can do this. And in both cases the set of links returned in the representation will depend upon the state of the resource and upon the requesting user. There is nothing intrinsically different between personal and shared state resources in this regard; the real objection to exposing domain entities through REST would seem to lie elsewhere.

---

<sup>35</sup> <http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven#comment-744>.

<sup>36</sup> And they argue quite strongly; see for example <http://java.dzone.com/articles/domain-model-rest-anti-pattern>.

---

### 29.4.1 Inductive vs Deductive Style

---

A more useful distinction is between systems that use an inductive style and those that use a deductive style. The inductive style is about taking the user through a series of steps in order to accomplish a goal. The inductive style works well when the users needs explicit assistance in order to navigate it. One of the earliest examples was Microsoft Money 2000<sup>37</sup>, which took users through various common-place financial book-keeping tasks.

The opposite of the inductive style is the deductive style, and a good example is a word-processor that starts with a blank page and more-or-less leaves the user to write the document in any order that they choose.

Other examples of deductive style apps are IDEs and the UNIX shell. The developer is free to write code in any order, or to string UNIX commands together as they see wish. Deductive style applications have much in common with sovereign applications<sup>38</sup>.

---

### 29.4.2 Application styles and Resource state

---

The key distinction between inductive and deductive style is about who is in charge – the user or the computer?. With an inductive application the process is hard-wired into the system, and the user must follow this process. With a deductive application the system offers the functionality to allow the user to accomplish their goal, but does not mandate the order of the user's interactions; the process is in the user's head - though there will almost certainly be rules implemented within the domain model to prevent actions that would be illegal or illogical given the current state.

There is no right or wrong to this; as already noted it depends on the experience of the user with respect to the domain. An inductive system can be frustrating to use for an experienced user, while a deductive system can leave an inexperienced user at a loss as to how to proceed.

Tying the above back to REST, applications built in the inductive style make heavy use of resources with personal state, with those resources modelling a user's goal and holding the state of the user's progression to that goal. The resource represents a use case instance, and its representation has links that represent the state transitions of the use case instance. These resources will most likely interact with underlying domain entities but those entities are never exposed.

In contrast, applications built in the deductive style will more likely make use of resources with shared state (domain entities), with the functionality

---

<sup>37</sup> <http://msdn.microsoft.com/en-us/library/ms997506.aspx>

<sup>38</sup> [http://en.wikipedia.org/wiki/Application\\_posture](http://en.wikipedia.org/wiki/Application_posture)

of those entities made directly available for the user to invoke as they see fit. This should not be confused with a simple CRUD system; the behaviour on the entities can be every bit as rich as the behaviour exposed by a use case resource.

Some systems provide a mix of inductive and deductive styles, with corresponding resources to match. In an internet shop, the browsing of the shop is deductive in nature; the user can hop from product to product as they see fit. The checkout process though is more well-defined, and users tend to expect to be taken through it in an inductive style.

A related approach is to start with a deductive system, and then to look for the "commonly-trodden path". These paths can be determined by observing experienced users' behaviour of the system, and then using this to provide inductive guidance for less experienced users.



# 30 DEALING WITH UNTRUSTED CLIENTS

*How implementations can avoid important information leaking out to untrusted clients*

## 30.1 Securing HREFs

One possible deployment for Restful Objects systems is a Restful Objects server on the internet, serving up JSON to a JavaScript "single-page app". In these cases, the href json-properties in the links will provide information about the domain objects that an untrusted client might use to hack the system.

For example, suppose that an end-user uses a domain service to look up their Customer details. This might result in a href:

```
http://~/objects/cus/12345
```

A malicious user might therefore try to browse to a related customer, e.g. <http://~/objects/CUS/12346> and from that uncover useful information. This is clearly not desirable.

While the Restful Objects spec does define the format of URLs, the format of the instance identifier part ("12345") is implementation-specific and therefore opaque. An implementation can protect itself from hacking by ensuring that the instance id that is served up within hrefs is encrypted using a private key. For additional security, this private key could even be re-generated either each time the server is restarted or even per session; this would allow the URL that identifies an object to change over time.

Another scenario is that other clients on the network could be snooping for valid resource URLs. The standard mechanism to address this risk is to deploy the application over SSL (https protocol).

## 30.2 Avoiding accidental traversals

Under Restful Objects, a domain object's properties, collections and actions can be disabled or even hidden. Implementations are expected to manage this through a system of end-user roles and permissions.

For implementations that work this way, care will need to be taken to ensure that an end-user does not have accidental permissions to a property, collection or action that allows them to gain access to other parts of the object graph. Implementations are expected to provide their own advice as to how to ensure this.





## 31 CLIENT VS SERVER EVOLUTION

*When content negotiation is or is not needed.*

### 31.1 Motivation

One of the goals of REST is to enable both the client and server to evolve independently. In particular, a server should be able to be enhanced to offer new functionality but existing clients should carry on working and simply ignore (be unaware of) this new functionality. As clients require the new functionality, they can be updated and deployed separately from the server.

In many cases it is relatively easy to maintain backward compatibility. For example, a representation of a Customer may initially include the Customer's firstName and lastName. Adding a new middleInitial property will not break existing clients.

Sometimes, however, representations are restructured in a way that *would* break the client. For example, the developer might want to restructure the representation of a Customer such that the firstName, middleInitial and lastName are all moved onto an associated Name object. In this case any client expecting there to be a firstName or lastName property on Customer will break.

### 31.2 Content Negotiation

The standard solution is to version representations, using the media type as the identifier for a particular version of a representation. It can however be difficult<sup>39</sup> to manage multiple versions of persisted entities (v1.Customer, v2.Customer etc).

A simpler approach is to use addressable view models §A2.2 to provide a versioned abstraction layer on top of the domain entities. Thus, one client may consume v2.CustomerViewModel, while another may consume v3.CustomerViewModel. Each of these view models delegates to an underlying Customer entity, but doesn't expose it directly.

Restful Objects does not mandate how implementations support addressable view models, but a code sketch is described in § **Error!**  
**Reference source not found..**

---

<sup>39</sup> Though possible; one approach is to introduce an abstraction layer in the persistence store; modern RDBMS support updateable views, for example.

Within HTTP the client indicates the version of the representation it requires using the **Accept** header. The server either serves up that representation with a matching **Content-Type** header, or returns a 406 "not acceptable" error. The name for this process is 'content negotiation' (sometimes shortened to 'conneg').

Restful Objects currently provides some of the infrastructure for conneg, through the "x-ro-domain-type" media type parameter §A2.4.2. Using this parameter the server indicates the domain type of each object representation it serves; this applies to all representations, be they of view models or of entities.

However full content negotiation is not currently supported in the spec; the client cannot specify the "x-ro-domain-type" media parameter in the **Accept** header in order to request a specific representation (or if they do, it will be ignored). A future version of the spec is likely to include support, however; see §34.1.

### 31.3 When Conneg isn't required

While content negotiation and view models are undoubtedly important, they do bring with them a substantial maintenance overhead for the application developer. It's therefore worth understanding when they are not needed.

First, if new versions of client(s) and server can be deployed at the same time (typically when the developer "owns both ends of the pipe"), then versioned media types can be dispensed with. This is a realistic scenario for many enterprise applications that are only used internally within organizations. It can also apply to external-facing applications where the client is deployed automatically. A growing form of this is 'single page web apps' where the client logic is written in JavaScript that may be readily refreshed from the server.

Another scenario where conneg is not required is when the client is generic, in other words able to process any representation served up by Restful Objects. Indeed, one of the objectives of Restful Objects is to enable the development of such generic clients.

The analogy here is the humble web browser that has built-in knowledge of the *text/html* media type and is able to render any representation with this media type. Similarly, a Restful Objects "browser" would have built in knowledge of the various representations served up by a server, and would use the "profile" parameter to render domain objects, lists, action prompts and so on.

## Restful Objects

---

To summarize,

Client/server dependency	Developed together	Independently developed
Client genericity		
Generic	no conneg	no conneg
Bespoke/custom	no conneg	conneg required

## 32 CODE SKETCH TO SUPPORT ADDRESSABLE VIEW MODELS

*How framework implementations might define an API to allow developers to support view models for conneg and HATEOAS*

### 32.1 Making View Models Addressable

As discussed in §A2.2, an addressable view model is a view model that, like a persistent domain entity, has an addressable resource, enabling it to expose actions.

This implies that it has an object identifier and some sort of server-side state. However, unlike a persistent domain entity, there is no database row, and so its state must be recreated on the fly.

There are several ways that an implementation of Restful Objects could do this, but a simple solution is to derive all state from the instance identifier of the resource's URL.

For example, the representation of a CustomerViewModel could contain the following link to an action:

```
http://~/objects/CVM/123/actions/findRecentOrders
```

Here the domainType is "CVM", and the instanceId is "123". The implementation will determine from its metamodel that the domainType "CVM" corresponds to CustomerViewModel. Next, the implementation could instantiate the view model and initialize it with the value of the instance identifier string "123".

The new view model in turn would use parse this instance identifier in order to recreate the rest of its state. For example, "123" might correspond to a (persistent) Customer object with id=123; the CustomerViewModel could look this up using an injected CustomerRepository domain service.

What about the creation of the instance identifier in the first place? Well, this could be the same process, but in reverse, with the view model providing the value. Both the setting and getting of the oid could be defined by the interface.

Putting the above together, in Java this would be:

```
public interface AddressableViewModel {  
    String getInstanceIdentifier();  
    void setInstanceIdentifier(String instanceIdentifier);  
}
```

where:

- `getInstanceIdentifier()` is used by the implementation to manufacture URLs in the representations
- `setInstanceIdentifier()` is used by the view model to recreate its state

The implementation of the view model would then be something like:

```
public class CustomerviewModel implements AddressableviewModel {  
    String getInstanceIdentifier() {  
        return "" + getCustomer().getId();  
    }  
    public void setInstanceIdentifier(String instanceIdentifier) {  
        int customerId = Integer.parseInt(instanceIdentifier);  
        setCustomer(customerRepository.findById(customerId));  
    }  
    ...  
    @Hidden  
    public Customer getCustomer() { ... }  
    public void setCustomer(Customer customer) { ... }  
    // CustomerRepository injected  
    ...  
}
```

From the developers' perspective, if the intent of using a view model is to provide a stable API for clients (as exposed by the "x-ro-domain-type" media-type parameter §A2.4.2), care should be taken to:

- ensure that the view model is versioned (to allow breaking changes to be introduced over time), and
- ensure that its representation never "leaks" out non-versioned properties.

The first can be accomplished by using a version number in the package/namespace, eg:

```
com.mycompany.myapp.viewmodels.customer.v2.CustomerviewModel
```

The second can be accomplished by ensuring that all visible properties are either scalar types §A2.5, or are other versioned view models. Any references to domain entities (such as `Customer`, above) should be hidden.



## 33 FAQs

### 33.1 "The spec defines URLs with a single instance identifier component. For objects that notionally have a 'composite primary key' (e.g. order line item) I'd rather separate out those parts out, (e.g. "objects/ORI/123/1" rather than "/objects/ORI/123-1/...") Can I change the URL format in this way?"

The short answer is: no, you can't change the URL format of the instance identifier. The slightly longer answer is: and you shouldn't care.

The primary reason that the specification does not allow the format of its URLs to be changed is to ensure that **all** the resources for **all** objects can be addressed in a uniform way. For example, the templated URL:

`http://~/objects/{domainType}/{instanceId}/properties/{propertyName}`

defines how to access any given property '*propertyName*' of any given object identified by its '*domainType*' and '*instanceId*'.

If instead the specification allowed an Order to be accessed as `/objects/ORD/123`, while an order line item to be accessed as, say, `/objects/ORI/123/1`, then this would break the above templating rules.

Restful Objects stance here echos one of the Berners-Lee's axioms of web architecture<sup>40</sup>: that URLs should be opaque. Because the spec fully supports HATEOAS, it is certainly possible for URLs to be treated as entirely opaque. Or, they can be treated as semi-opaque: the instance identifiers are opaque, the rest of URL is as defined by its template.

One other benefit of this approach is that implementations are free to encrypt instance ids. This could be used to prevent a rogue client from generating URLs that would give it access to restricted resources.

### 33.2 "Why doesn't Restful Objects define its own media-types?"

Many REST practitioners recommend minting custom media types for every representation. Such media types effectively encode the semantics of the representations.

---

<sup>40</sup> Tim Berners-Lee, <http://www.w3.org/DesignIssues/Axioms.html>

If Restful Objects had taken this approach, it would have defined media types such as:

```
application/vnd.org.restfulobjects.object+json
```

and

```
application/vnd.org.restfulobjects.property+json
```

instead of:

```
application/json;  
  profile="org.restfulobjects/object";  
  x-ro-domain-type=com.mycompany.myapp.Customer
```

and

```
application/json;profile="org.restfulobjects/property"
```

However, there are two big issues with vendor-specific media types.

The first is that developer tools (e.g. the JSONView or RESTConsole plugin for Chrome web browser) do not understand these media types, and so make it hard for a developer to informally browse the representations.

The second is that the media type only defines one level of abstraction: "application/vnd.org.restfulobjects.object+json" indicates a representation of a domain object, but does not indicate its type. In contrast, the media type defined by Restful Objects, through its use of media type parameters, defines three levels of abstraction: that the representation is JSON, that it is a domain object, and that it is a domain object of a certain type.

It would, of course, be possible to add a media type parameter x-ro-domain-type to a custom media type, but why bother? Better to overlay the additional layers of abstraction.

### 33.3 "Restful Objects can expose domain entities as resources. But doesn't exposing domain entities mean that the server and client are closely coupled, thereby violating REST?"

To answer this properly we should distinguish three cases.

The first is to note that in many cases the server and client will in fact be closely coupled. For example, in many internal enterprise applications both will be developed and deployed by the same team. For such scenarios the coupling described is of no concern.

For cases where the client is independently developed from the server, we should distinguish generic clients from bespoke clients. The media types for the representations defined by Restful Objects have been carefully designed such that a fully generic client could be written, driven purely from the hypertext.



Such an application would be an example of the Naked Objects pattern (it could use the "x-ro-domain-type" and "x-ro-element-type" media parameters to customize the display of certain objects and collections). While Naked Objects is not a common architecture, it is worth noting the similarity with web browsers; a web browser can consume any text/html and render it to a user. A generic Restful Objects application would in effect be doing the same thing.

The last case to consider is an independently-developed client that is bespoke; in other words it makes hard-coded assumptions about the structure of the JSON representations.

For these cases, it is currently the case that Restful Objects offers limited support; the domain type of the representation is advertised through the "x-ro-domain-type" media type parameter of the **Content-Type** header, but it is not possible for the client to request a particular representation by setting the Accept header. This is likely to be added as a future enhancement, see §34.1.

### 33.4 But isn't exposing domain entities just the wrong thing to do? Surely I should be exposing use cases as resources?

Not necessarily; both are valid approaches.

Some practitioners have claimed that exposing entities makes it difficult to render the relevant links to support hypermedia-driven designs. While this may be a legitimate issue with some frameworks, such issues are not a problem with Restful Objects (due to framework implementations' use of an underlying metamodel).

As noted in §29, Restful Objects is agnostic as to what it exposes: use cases, or entities. Because of that, you can start off exposing behaviour directly on entities. Later on – if you find that you need them – then you can start to add in either commands/use case objects to support the commonly-trod paths.

In other words, Restful Objects lets you add in layering as and when it's justified, but allows you to defer that decision until a later point in the project when you've learnt more.

**33.5 I prefer having an application service layer that exposes use cases and views. Doesn't Restful Objects simply move the design work that I would normally have done in my resources back a level".**

Yes, it does.

But the benefits of doing so are that:

- it reduces the learning curve for new developers
- it separates responsibilities of your system
- it eliminates the need to explicitly document the REST API, and
- it makes the domain classes easier/faster to test.

There's further discussion on all this in §A1.3.

## 34 IDEAS FOR FUTURE EXTENSIONS TO THE SPECIFICATION

*Ideas for extending the scope of the specification*

### 34.1 Content Negotiation

Although representations of domain objects indicate the type of the object in the **Content-Type** header (through the **x-ro-domain-type** media parameter §A2.4.2), this parameter is ignored when set on the **Accept** header. This means that a client cannot insist that a particular representation is of a certain type.

This idea is for the spec to be extended so that the **x-ro-domain-type** from an **Accept** header is checked when invoking an object or service action, and is used as a hint to ensure that a representation of the correct type is returned to the client. In this way the spec would support content negotiation (conneg).

In theory, this functionality could be applied to any domain type, either a persistent entity or an (addressable) view model. In practice, though, supporting different versions of persistent entities (v2.Customer, v3.Customer) may well be difficult, and so implementations may choose to restrict support to actions that return addressable view models, §32.1.

For example, given a versioned CustomerViewModel, a client would set the **Accept** request header to:

```
Accept: application/json;
       profile="urn:org.restfulobjects:repr-types/object";
       x-ro-domain-type=
         "http://~/domain-types/x.viewmodels.v2.CustomerViewModel"
```

The server would then either serve up a representation with a matching Content-Type, or would return a 406 error code ("Not acceptable") to indicate that the domain type required is not (or is no longer) supported.

It is easy enough for the framework implementation to parse the **x-ro-domain-type** and determine the corresponding domain type (e.g. java.lang.Class on a Java implementation, or System.Type on a .NET implementation). The remaining question is how to ensure that this required type is returned?

## *Domain Model Agnostic*

One approach could be for the domain model to remain ignorant of the required return type, and for the framework implementation to instead define an API that allows converters to be registered. These would be responsible for preserving backward compatibility, manufacturing previous versions of domain types as required:

```
public interface Converter {  
    public <F, T> T convert(F from, Class<T> to);  
}
```

For example, suppose the **Accept** header requests that a v2.CustomerViewModel be returned, but the action invoked is on a Customer that always returns the current (in this case v3) CustomerViewModel:

```
public class Customer {  
    ...  
    x.v3.CustomerViewModel summarize() { ... }  
}
```

Because the representation of this returned object would be incompatible with the requested type, the framework instead looks for a registered converter:

```
Class<?> requiredReturnType;  
Object objectToReturn = ...;  
  
Converter converter = converterRegistry.find(  
    objectToReturn.getClass(), requiredReturnType);  
  
if (converter == null) { ... throw a 406 ... }  
return converter.convert(objectToReturn, requiredReturnType);
```

## *Domain Model Aware*

An alternative design is for the domain object to be told which implementation of the view model to return. In .NET, for example, this could be done with a generic type:

```
public class Customer {  
    ...  
    public T summarize<T>() where T: ViewModel { ... }  
}
```

Here the framework could reflectively invoke the method with the appropriate value for T as determined from the **x-ro-domain-type** parameter. The method body could use this type parameter (eg in a switch statement) to create and return an object of the appropriate type.

## 34.2 Sorting (x-ro-sort-by)

This suggestion is for the Restful Objects spec to define the capability (probably optional §B8) to allow sorting of returned lists §B11 or object collections §C17.5.

If supported, the order in which links are returned within the list may be influenced using a reserved **x-ro-sort-by** query param. If present, this parameter would specify a comma separated list of sort properties, indicating ascending or descending for each (similar to an ORDER BY statement in SQL).

For example, for a resource returning a list of links to Customers, setting **x-ro-sort-by** to:

```
mostRecentOrder.placedOn desc, lastName, firstName
```

would order those links by the Customer's mostRecentOrder's placedOn date in descending order, then by the Customer's lastName ascending, then by firstName ascending. Note that multipart property keys could be supported (that is: ordering is not on a direct property of Customer, it is on the property of an Order which is in turn one of the properties of Customer).

To indicate that sorting has occurred, the representation would include the **"sortedBy"** json-property. This would contain the original requested value, along with the value in a "normalized" form. For example:

```
{
  "sortedBy": {
    "requested":
      "mostRecentOrder.placedOn desc, lastName, firstName",
    "normalized": [{
      "clause": "mostRecentOrder.placedOn",
      "direction": "desc"
    }, {
      "clause": "lastName",
      "direction": "asc"
    }, {
      "clause": "firstName",
      "direction": "asc"
    },
    ...
  ]
}.
"value": [
  ...
]
```

Note that the **"sortedBy"** json-property would need to be a list (rather than a map) because the order of keys in a JSON map is not guaranteed.

### 34.3 Pagination (x-ro-page, x-ro-page-size)

This suggestion is for the Restful Objects specification to define the capability (probably optional §B8) to allow object lists §B11 (as returned from action invocations) to be paginated.

If supported, the client could optionally request that a returned list be paginated, by setting a reserved **x-ro-page** query parameter to specify which page of objects is being requested, and a **x-ro-page-size** query parameter to specify the size of each page.

For example:

- `x-ro-page=3&x-ro-page-size=25`

would specify returning a representation for objects 51~75 in the list.

To indicate which page set has been returned, the representation would include a **"pagination"** json-property, which has the requested **"page"** and **"pageSize"** json-properties. It would also include the **"numPages"** for the specified page size, as well as the **"totalCount"**. In addition, the representation would provide a **"links"** json-property that has links to the `rel=previous` and `rel=next` pages.

For example:

```
{
  ...
  "pagination": {
    "page": 3,
    "pageSize": 25,
    "numPages": 4,
    "totalCount": 82,
    "links": [ {
      "rel": "previous",
      "href": ...,
      "type": ...,
    }, {
      "rel": "next",
      "href": ...,
      "type": ...,
    }
  ]
}
"value": [
  ...
]
```

Using this information the client could manage the paging, for example enabling/disabling *next* and *previous* buttons in its UI.

### 34.4 Minimizing Round-trips (x-ro-follow-links)

While HTTP caching §A2.13 works well enough for non-transactional resources, most of the resources served up by Restful Objects will be transactional. This suggestion is for the Restful Objects spec to define a capability (probably optional §B8) to support 'eager following' of links.

This capability would be specified by setting a reserved **x-ro-follow-links** query parameter. This would act as a hint to the server to generate in its response a representation that includes additional information as a result of following links.

## Restful Objects

---

For example, the client could use this query parameter to:

- obtain additional property details for the object resource, eg, to support an "object edit" use case
- obtain details of objects referenced in a collection, eg, to support rendering the collection in table view format

The query argument would typically be a semi-colon separated list of strings, each element being the json-property of a link within the representation to be followed.

For example, the domain object representation §C14.4 has links to each member of the object:

```
"members": {
  "createdOn": {
    "memberType": "property",
    "value": ...,
    "links": [ {
      "rel": ".../details;property=\"createdOn\"",
      "href": "...",
      ...
    }, ... ]
  },
  "customer": {
    "memberType": "property",
    "value": ...,
    "links": [ {
      "rel": ".../details;property=\"customer\"",
      "href": "...",
      ...
    }, ... ]
  },
  "items": {
    "memberType": "collection",
    "links": [ {
      "rel": ".../details;collection=\"items\"",
      "href": "...",
      ...
    }, ... ]
  },
  "confirm": {
    "memberType": "action",
    "links": [ {
      "rel": ".../details;action=\"confirm\"",
      "href": "...",
      ...
    }, ... ]
  },
  ...
}
]
```

A common use for the proposed **x-ro-follow-links** would be to request the population of a "value" json-property for any node in the map. For example:

- members.items  
would populate the "value" json-property of the items collection.

- `members[memberType=property].links[rel=urn:org.restfulobjects:rels/details]`

would follow the "details" link of every object property

- `members.confirm.links[rel=urn:org.restfulobjects:rels/details]`

would follow the details link of the `confirm()` action

In all these cases the identified elements are links; the returned representation would include a **"value"** json-property for the identified links.

As an alternative to using paths, the **x-ro-follow-links** could specify a well-defined ("precanned") value that is defined by that resource. For example, the GET Object resource §C14.1 could define "ObjectEdit" as a hint to additionally include property details.

If the parameter were present and contained a value that did not represent a link or were otherwise not understood by the server, then the server would silently ignore the query parameter.

The **x-ro-follow-links** query parameter could also be used to influence the loading of collections:

- setting the query parameter to **"links[rel=.../details]"** could cause the details link to be populated, from which full information about the contents of the collection can be obtained;
- setting the query parameter to **"value"** could cause the optional **"value"** to be returned, holding a list of links to the actual elements. These links would have their **"title"** json-property §A4.1 populated;
- setting the query parameter to **"size"** could cause the optional **"size"** to be returned. This is useful if the client needs to know only the number of elements in a collection.

These three values for **x-ro-follow-links** should be considered as mutually exclusive (since: details => value => size).

From the client's perspective, note that this means that the contents of the collection would be available either in the **"value"** json-property, or could be in the in-lined details representation **"links[rel=.../details].value"** json-property.

### 34.5 Partial Arguments

This suggestion is for the Restful Objects specification to define support for partial arguments. This would probably be an optional capability §B8.



The idea is that the action resource §C18 would accept a partial argument map, and use this to tailor the choices available for other, non-constrained arguments. The main use case is for actions that take multiple parameters where the valid choices for one parameter depend on the value of another parameter; for example category/subcategory or country/region.

In the example introduced above, if an object has an action `listProducts(Category category, Subcategory subcategory)`, then the following partial argument maps could be provided as the query string to the action resource:

```
{
  "category": {
    "value": {
      "href": "http://~/objects/CGY-BOOKS"
    }
  }
}
```

The returned response would restrict the "**choices**" json-property of the subcategory to be those relevant for the category of books:

```
{
  "category": {
    "value": {
      "href": "http://~/objects/CGY/BOOKS"
    }
  },
  "subcategory": {
    "choices": [
      { "href": "http://~/objects/SCY/Fiction" },
      { "href": "http://~/objects/SCY/Childrens" },
      { "href": "http://~/objects/SCY/Computer" },
      { "href": "http://~/objects/SCY/Business" }
    ]
  }
}
```

### *Validating argument sets*

The client can also request the validation of arguments; this is done by providing the reserved **x-ro-validate-only** param (§A3.2)<sup>41</sup>.

For example, to validate the category by itself (for example, when the user tabs from the category field in the UI), it would provide only the category argument:

---

<sup>41</sup> The "x-ro-" prefix is used to distinguish from regular argument names.

```
{
  "category": {
    "value": {
      "href": "http://~/objects/CGY/BOOK"
    }
  },
  "x-ro-validate-only": true
}
```

If the server found that the argument provided was invalid, then it would indicate it in its response using the **"invalidReason"** json-property:

```
{
  "category": {
    "value": {
      "href": "http://~/objects/CGY/BOOK"
    },
    "invalidReason": "not permitted to select from this category "
  }
}
```

### 34.6 Internationalisation

This suggestion is for the Restful Objects specification to define support for internationalization. This would probably be an optional capability §B8.

The Restful Objects spec could support internationalization as follows:

- json-property keys in representations are never internationalized
- json-property values for selected keys are internationalized; and these are explicitly identified in the spec detail.
- Internationalized values would be with respect to the **Accept-Language** HTTP header.
- Broadly speaking, those json-properties that are internationalized either represent "friendly" names, or descriptions, or are invalidity/disabled reasons.
- The json-properties that are internationalized will only ever be simple strings (with a "format" of "string", §A2.5). Strings with other formats (e.g. decimal numbers, or dates) are never internationalised.

### 34.7 Listable Instances

This suggestion is to allow the `~/objects/{domainType}` resource to support the GET method. Doing so would return all instances of that type, as a list representation §B11.

For example,

```
~/objects/ORS
```

might return all instances of the OrderStatus class

Not every domain type is likely to be listable; it wouldn't be feasible or desirable to return a representation for a type that has millions of instances. Therefore the domain type representation §D22 would indicate whether a type is "**listable**" (as a new json-property). Instances that are not listable would return a 405.

### 34.8 Addressable Parent Resources

Although URLs should be considered opaque, nevertheless there is often an expectation that for any given URL, all parent URLs are defined.

This is not currently the case with Restful Objects, as there are no definitions for resources that represent all members of a certain member type:

- `~/objects/{domainType}`
  - except for POST; see also §34.7.
- `~/objects/{domainType}/{instanceId}/properties`
- `~/objects/{domainType}/{instanceId}/collections`
- `~/objects/{domainType}/{instanceId}/actions`
- `~/services/{serviceId}/actions`
- `~/domain-types/{domainType}/properties`
- `~/domain-types/{domainType}/collections`
- `~/domain-types/{domainType}/actions`

One obvious definition for these resources is to be a subset of the parent object or domainType resource, restricted to the member type in question.

For example,

```
~/objects/{domainType}/{instanceId}/properties
```

could return the same representation as

```
~/objects/{domainType}/{instanceId}
```

except that only the properties would be included in the "**members**" list.

Another simpler option might be to define these resources as returning a 303 "See Other", in effect redirecting the client to the parent object or domainType resource.

### 34.9 See other for action-results

Currently the action-results representation §C19.4 can return an in-lined domain object. This is intended to be a convenience; the **ETag** header is suppressed.

## Restful Objects

---

An alternative design<sup>42</sup> would be to have the action-result return a 303 "see other" in this situation, and include a reference to the object.

The desired behaviour could be made tunable, akin to the optional capability that the spec provides for domain model schemes.

The "**actionResult**" optional capability would return:

- "in-line"
  - return a representation of the domain object in-line
  - ie the current behaviour
- "seeOther"
  - return a 303 response to the returned domain object
  - ie the behaviour suggested above
- "selectable"
  - as requested by the client

If the last option were supported, the client could then use a new "**x-ro-action-result**" query parameter to indicate its preference:

- "in-line"
- "seeOther"

If not specified, then the default would be "in-line".

---

<sup>42</sup> As recommended by Masse, REST API Design Rule book \*\*\*.