Restful Objects

Draft v0.61

Restful Objects defines a set of RESTful resources, and corresponding 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 where you will also find details of known implementations, and other useful information.

Dan Haywood

TABLE OF CONTENTS

TABLE OF FIGURES

A

CONCEPTS AND BUILDING BLOCKS

1 INTRODUCTION

Restful Objects defines a set of RESTful[1] resources[2] 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 (eg 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 clients to be written fully generically. Such clients would be similar to a web browser that can consume any HTML.

Alternatively, the developer may write custom clients that have some shared knowledge of the domain being exposed, and render the information in a more specific fashion. The analogy here is a screen-scraping application that looks for certain fragments of HTML in order to create a mash-up.

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 another entity type;
- 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 resources.

[1] http://en.wikipedia.org/wiki/REST

[2] 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 standard HTTP methods used (GET, POST, PUT, DELETE) to call the resource URLs;
- the format of URLs used to access the domain object resources (though URLs can also be treated as opaque);
- the standard HTTP headers supported for both request and response;
- the use of standard HTTP status return codes;
- support for content negotiation through parameterized media types e.g. application/json;profile=urn:org.restfulobjects /profiles/object;domainType=urn: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 (eg validation, paging, sorting).

Existing HTTP standards and supporting W3C standards have been used wherever possible[3].

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.

[3] The spec tries not to reinvent existing wheels, but we always welcome suggestions as to other existing standards that Restful Objects might usefully incorporate.

- **Documentation**. Restful Objects eliminates the need for the project team to explicitly document the REST API, because consumers can infer the structure either directly from the underlying domain classes, or by querying the domain metamodel resources §D.
- **Frameworks**, because Restful Objects is a spec, it also for frameworks to be written that implement the spec. 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.
- **Clients**. 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, "Patterns and Practices".

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[4];
- for developers intending to write a generic or custom client against a domain object model exposed using Restful Objects;
- and 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

The original idea for this specification and primary author is Dan Haywood. Substantial contributions have been made by Richard Pawson and Stef Cascarini.

[4] A list of known implementations of the Restful Objects specification is maintained at www.restfulobjects.org

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. Again, this has been shown by an "x" placeholder: x.Customer.

There are two important concepts in the spec that both have the name "property": the domain object property (or field), 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".

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

[5] http://creativecommons.org/licenses/by-sa/3.0/

Version	Description
v0.42	• replaced 403 (forbidden) with 401 (not authorized) Following feedback from REST community: • Additional discussion on the goals of the spec, plus FAQ • Removed custom X-Representation-Type response header, instead use application/vnd.ro-xxx+json media type (based on Accept header); • 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.) • added "next" and "previous" links for list representation so that paging is performed in a standard fashion • changed validation failure to return 400 rather than 412 • specify use of ETags for concurrency control rather than Last-Modified (lack of precision of Last-Modified); returns 412 rather than 409 • changed successful validation to return 204 rather than 304
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 inlined. 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).

Version	Description
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.
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.

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

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:

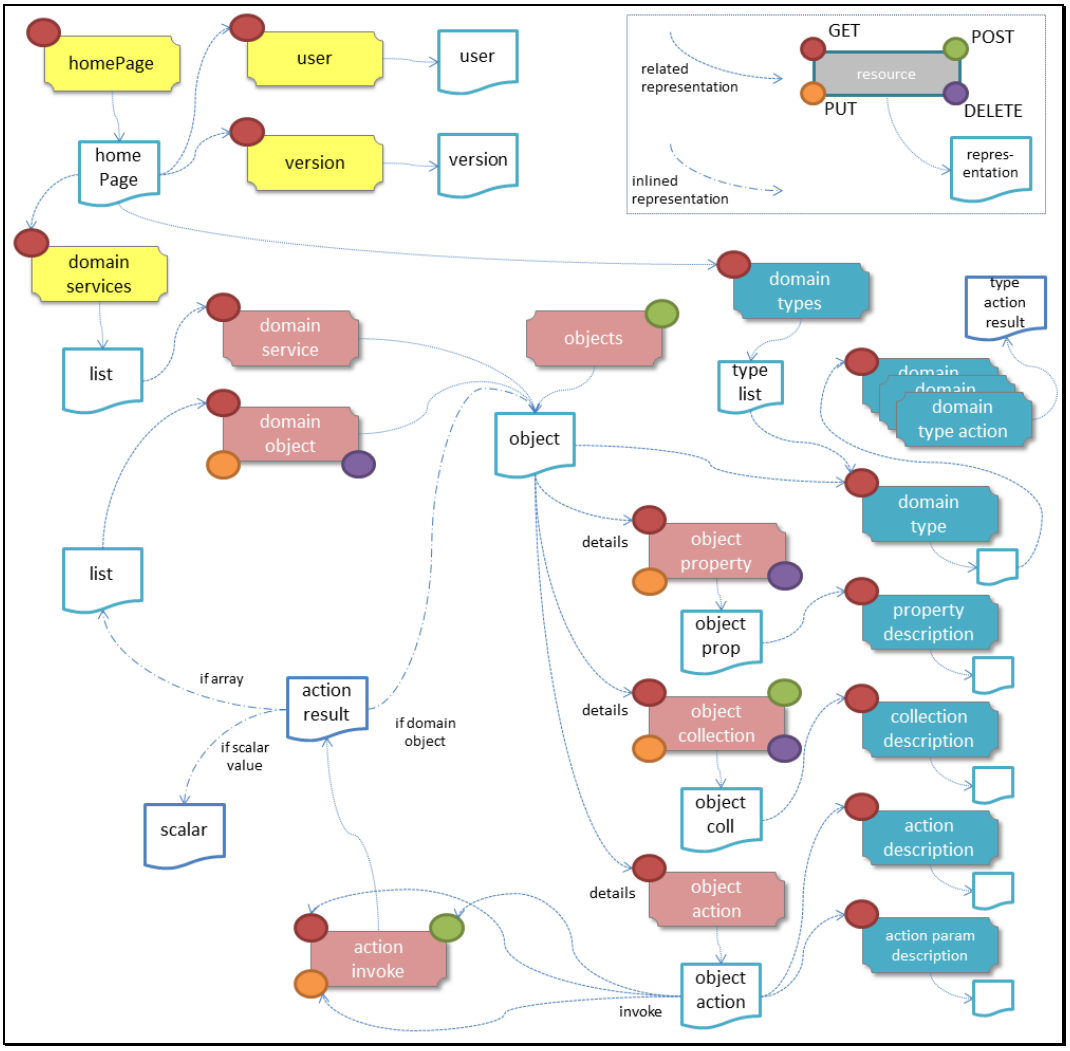

FIGURE 1: RESOURCES AND REPRESENTATIONS

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 model resources §D;
- The circles on 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 generated by the resource.
  Many resources generate their own, unique, representation (eg the home page resource, the user resource). However, some representations (most notably the object representation) may be generated by several different resources.
- Text on the arrow from a resource to a representation indicates the HTTP method that generates that representation (the default being GET);
- Dotted arrows 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;
- Inlined 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 of 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. Since each domain service is actually just a domain object, following the link generates in turn a domain object representation.
- 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 (eg whether it supports automatic pagination of results).

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).

From the (representations of these) sub-resources it is then 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.

Restful Objects defines two schemes by which implementations can 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 (§D20.2, §D21.2, §D22.2, §D23.2, §D24.2, and §D25.2).

Restful Objects also defines a general mechanism to enable new domain object instances to be created and persisted, through the Objects 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 §E27.1. And one could leave the discussion at that; either a link is present or it is not, and the client must act accordingly.

However, 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 §C18.1on 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.

In almost all cases a representation of a persistent domain entity includes links to the entity's state (its properties and collections), though in rare cases the values of this state may all be in-lined within the object's representation.

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.

Transient domain entity

A transient domain entity is an object instance whose creation is under the control of the client.

Unlike a persistent domain entity, for a transient domain entity there is no server-side resource to address. This means that the representation of a transient domain entity 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 a transient object 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 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 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 are different: the server must ensure that any representations consumed by its clients remain backwardly compatible.

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

View models are not persisted and so (like transient entities) their representation includes the state of the view model but no behaviour. However, unlike transient entities, they provide no persist link. In fact, such representations contain 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 models 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 recreate some server-state. See §E31.1 for a code sketch as to how this might be accomplished.

In theory actionable 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 inline the property or collection values in their representation.

A good example of an actionable 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 in the object representation §C14 to other representations:

	property	collection	action	persist	update properties	delete
Persistent entity	yes (usually)	yes (usually)	yes	---	yes	yes
Transient 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 or a link, the spec uses the above terms as a way to explain why that variation occurs.

2.3 Domain Object Resources

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

	§B9 Objects/	§C14 Objects/ {Object}	§C14.4 Objects/ {Object}/ Properties/ {Property}	§C16.4 Objects/ {Object}/ Collections/ {Collection}	§C17.5 Objects/ {Object} Actions/ {Action}	§C18.2 Objects/ {Object}/ Actions/ {Action}/ invoke
GET	create transient instance	object summary, member summary, property values	property details and value	collection details and content	action prompt	invoke (if 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 (if idempotent)
DELETE	n/a – 405	delete object	clear value	remove object	n/a – 405	n/a – 405
POST	persist transient instance	n/a – 405	n/a - 405	add object (if list semantics)	n/a – 405	invoke (any)

The columns indicate the domain object resources shown in the Figure 1, plus the Objects resource §B9 used for creating and persisting new object instances. Meanwhile the rows define the HTTP methods 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 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[6], 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.

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 it 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.

Finally, for actions the actual invocation of an action is accomplished 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[7], 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[8].

---

[6] 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.

[7] 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). A query only action is always idempotent)

[8] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.7

2.3.1 Example Resource URLs

The following table lists some example URLs for accessing resources:

Resource Type	Resource
object	http://~/objects/ORD-123
property	http://~/objects/ORD-123/properties/createdOn
collection	http://~/objects/ORD-123/collections/items
action	http://~/objects/ORD-123/actions/placeOrder
action invocation	http://~/objects/ORD-123/actions/placeOrder/invoke

In the example URLs the "ORD-123" is an object identifier to a persisted instance of a domain object (an Order, in this case). The format of this object identifier is implementation-specific, though that it must be URL-encoded.

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	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
Follow link to 'Find By Name' action (to display to user as a dialog)	GET	http://~/services/x.ProductRepository/actions/FindByName	-	Action
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 a Product

Description	Method	URL	Body	Returned representation
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/FindByName/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/object/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.

The standard media type for JSON representations is *application/json*[9]. However, this says nothing about the semantics of that data; it's analogous to returning some XML without indicating the XML schema/DTD that the XML complies with.

Restful Objects therefore uses two media type parameters to provide additional higher-level semantics.

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[10].

[9] Note that the spec also supports for non-JSON media types, such as application/pdf, for blobs and clobs (aka attachments). See § A3.6.

[10] http://buzzword.org.uk/2009/draft-inkster-profile-parameter-00.html

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/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
list	a list of references to domain objects/services §B11
value	a scalar value (as opposed to an entity), for example as inlined within an action invocation's result §C19.4
object	a domain object instance (which may represent an entity or a service) §C14.4
objectproperty	a domain object property §C16.4
objectcollection	a domain object collection §C17.5
objectaction	a domain object action §C18.2
actionresult	result of invoking a domain object action §C19.4
typelist	a list of domain types §D20.2
domaintype	a domain type §D21.2
propertydescription	a domain property's description § D22.2
collectiondescription	a domain collection's description §D23.2
actiondescription	a domain action's description. §D24.2
actionparamdescription	an action parameter's description §D25.2
typeactionresult	result of invoking a domain type action §D26.
error	An error was generated, §B10.

2.4.2 Domain Type ("x-ro-domaintype" parameter)

While the "profile" parameter informs the client of the representation type, in the case of an object representation it does not distinguish between, for example, a Customer and an Order.

For clients that want to handle such representations differently, the spec defines an additional "x-ro-domaintype" parameter[11]. The value of this parameter is a URI to the domain type reference (see §3.1, and §D21).

For example, the media type for the representation of a Customer (persistent or transient entity) would be of the form:

```
application/json;profile="urn:org.restfulobjects/object";x-ro-
domaintype="http://~/types/com.mycompany.myapp.Customer"
```

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

```
application/json;profile="urn:org.restfulobjects/object";x-ro-
domaintype="http://~/types/com.mycompany.myapp.viewmodels.v2.OrderH
istory"
```

The client can use this value to process the representation accordingly; for example, rendering it with a different view template. It can also set the value of this parameter in the **Accept** header in order to request such a representation (as discussed immediately below, §2.4).

2.4.3 Handling of Accept headers

The HTTP protocol[12] 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 was 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" or "x-ro-domain-type" parameters will be ignored
- if the client provides an **Accept** header of application/json but omits the "profile" parameter or "x-ro-domain-type" in its **Accept** header, then the server may return any JSON representation.
  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.

[11] Unlike the "profile" parameter, there is no standard or semi-standard parameter to reuse. The "x-ro-" prefix of the "x-ro-domaintype" parameter is to avoid name clashes with other emerging standards.

[12] http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

- if the client specifies the "x-ro-domaintype" parameter, then a "profile" of urn:org.restfulobjects/object is assumed, and the returned representation must be of the same type.
  If it is not, then a 406 must be returned.

For example, a representation of a view model of type *x.viewmodels.v2.OrderHistory* could be returned based on an **Accept** header of:

```
application/json
```

or of:

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

or of:

```
application/json;profile="urn:org.restfulobjects/object";
  x-ro-domain-type="x.viewmodels.v2.OrderHistory"
```

However, a 406 would be returned if the **Accept** header were:

```
application/json;profile="urn:org.restfulobjects/list"
```

say, or indeed if it were:

```
application/json;profile="urn:org.restfulobjects/object";
  x-ro-domain-type="x.viewmodels.v1.OrderHistory"
```

(note the v1, not v2 in the domain type).

Restful Objects does not specify how a framework implementation should go about ensuring this behaviour. However, a code sketch presenting one possible approach is outlined in §E31.2.

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/object",
  application/json;profile="urn:org.restfulobjects/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).

Irrespective of what the client provides in its **Accept** header, the returned **Content-Type** will however always indicate the "profile" parameter and also the "**x-ro-domainType**" (if an object representation).

2.4.4 Browsing the RESTful API

During development it can be helpful to browse a RESTful API using a browser plugin such as RESTConsole or JSONView. Such plugins provide such features as folding of the JSON representation, and 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 do not set the **Accept** header to *application/json*. In order to accommodate the use of such tools, if the **Accept** header contains only the values normally set by browsers (eg text/html) then **Accept** header validation should be suppressed. Again, the **Content-Type** returned will be set to *application/json* along with the profile parameter.

2.5 Scalar datatypes and formats

JSON defines only the following scalar datatypes[13]:

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

The JSON schema specification[14] also defines:

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

Most notably, JSON does *not* define a native datatype to represent date, time or datetime. 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 a means of describing how to interpret the value of a string json-property, with the "format" json-property being used in the optional capability to describe domain metadata §3.1.

The values of the "**format**" json-property are[15]:

- 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)
- date-time
  - A date in ISO 8601 format of YYYY-MM-DDThh:mm:ssZ in UTC time.

[13] http://json.org/, also http://en.wikipedia.org/wiki/JSON#Data_types.2C_syntax_and_example

[14] http://tools.ietf.org/html/draft-zyp-json-schema-03, section 5.1.

[15] 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
  - 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.
- biginteger
  - The value should be parsed as a big integer.
- bigdecimal(n)
  - The value should be parsed as a big decimal, scale n.
- 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

If the implementation supports the formal metamodel scheme §3.1.2, then each of these datatypes has a corresponding domain type resource §D21 so that it can be linked to by properties, collections etc.

In the case of a blob or a clob, the "**value**" of the property/argument is suppressed from both the object representation §C14.4, §**Error! Reference source not found.**. Instead, it is available in the property representation §C16 either in-lined within that representation, or (if attachments are supported) as an attachment link. Attachments are an optional capability discussed further in §3.6.

2.6 Link representation

Every JSON representation may have relationships to other representations, and this 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": { ... }
}
```

where:

JSON-Property	Description
rel	A URN indicating the nature of the relationship of the related resource to the resource that generated this representation.

JSON-Property	Description
href	The (absolute) address of the related resource. Any characters that are invalid in URLs must be URL encoded.
type	The media type that the representation obtained by following the link will return.
method	The HTTP method to use to traverse the link (GET, POST, PUT or DELETE)
title	(optional) string that the consuming application may use to render the link without necessarily traversing the link in advance
arguments	(optional) map that may be used as the basis for any data (arguments or properties) required to follow the link. Discussed further below.
value	(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.

"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[16] or a value specific to Restful Objects:

rel	Specified by	Description
self	IANA	"Conveys an identifier for the link's context", in other words, following this link returns the same representation. Discussed further in §2.7.
describedby	IANA	"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
help	IANA	"Refers to context-sensitive help"
icon	IANA	"Refers to an icon representing the link's context." A scalable icon for any purpose
icon16	Restful Objects	A 16x16 pixel icon
icon32	Restful Objects	A 32x32 pixel icon

[16] http://www.iana.org/assignments/link-relations/link-relations.xml

rel	Specified by	Description
up	IANA	Link from member to parent object/type, or from action param to its action
object	Restful Objects	A domain object
service	Restful Objects	A domain service
choice	Restful Objects	A domain object acting as a choice for a property or an action parameter
default	Restful Objects	A domain object acting as a default for a property or an action parameter
details	Restful Objects	Details of a member of a domain object/service
modify	Restful Objects	Modify a domain object property or properties.
clear	Restful Objects	Clear a domain object property
addto	Restful Objects	Add to a domain object collection
removefrom	Restful Objects	Remove from a domain object collection
invoke	Restful Objects	Invoke a domain object action
persist	Restful Objects	Persist a transient object
property	Restful Objects	Linked from domain type to a property
collection	Restful Objects	Linked from domain type to a collection
action	Restful Objects	Linked from domain type to an action
actionparam	Restful Objects	A domain type param, linked from an action
returntype	Restful Objects	The domain type which represents the (return) type of a property, collection, action or param
elementtype	Restful Objects	The domain type which represents the element of a collection
version	Restful Objects	Version of the spec and implementation
user	Restful Objects	The current user
services	Restful Objects	The set of available domain services
domaintypes	Restful Objects	The domain types available in the system.
domaintype	Restful Objects	A domain type.
attachment	Restful Objects	An attachment for a property value; see 3.6.

"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 may (depending on the implementation §B8) have an additional "profile" parameter to further describe the representation; for example:

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

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

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

"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 §2.8.

"value"

JSON may be used to represent either a value type (eg String, date, int) or a reference type (eg 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 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",
  ...
}
```

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": "object",
    "href": "http://~/objects/ORS|IN_PROGRESS",
    "type": "application/json;profile=\".../object\"",
    "title": "In Progress",
    "method": "GET"
  },
  ...
}
```


2.7 "self"

The vast majority of representations include a "self" link, addressing 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=\".../objectproperty\"",
      "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 representations that do not have a "**self**" link.

The first set is representations of transient objects or of view models § 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.

2.8 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.

2.8.1 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 objects/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 if providing a set of arguments that represent object properties (if update multiple properties §C14.2). For these cases the 'Formal' mechanism must be used §3.1.2.

2.8.2 Formal Arguments

Although simple arguments §2.8.1 are convenient to use, their applicability is limited. For all other cases arguments[17] 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)

[17] 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.

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 do not support named parameters, the recommendation is to manufacture one either using existing metadata where available (eg 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 §2.9 explains the mechanics of how the argument structures defined here are passed to the resource.

2.8.2.1 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
- the argument can be omitted from the map to request either:
  - that validation should be performed only on those arguments that have been provided, or
  - that a representation is generated based on the values of those arguments that *have* been provided

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
value	is the value of the argument (possibly a link)
invalidReason	(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"
  }
}
```

2.8.2.2 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"
}
```

2.8.2.3 Argument maps (Actions, Properties)

Some resources, notably the action resources (§C19.2, §C19.3) but also 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.

For example, suppose an object has an action findProduct(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 be ignored.

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 inline within a map. In the case of a blob, the byte array must be base 64 encoded.

Validating individual property/arguments

In both cases, if any of the values provided are invalid, then (again) 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"
  }
}
```

2.8.2.4 Validating argument sets

The client can also request the validation of arguments; this is done by providing the reserved **x-ro-validate-only** param (§3.3)[18].

[18] The "x-ro-" prefix is used to distinguish from regular argument names.

In the example introduced above, an object has an action findProduct(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": "renamed to CGY-BOOKS"
  }
}
```

2.8.2.5 Obtaining argument choices

Continuing with the same example, it's likely that the set of subcategories will depend on the category that's been selected.

Provided that the server implementation supports this optional capability (§B8.2), the client can obtain these choices by submitting the following to the GET Action resource (§C18.1.1).

For example:

```
{
  "category": {
    "value": {
      "href": "http://~/objects/CGY-BOOKS"
    }
  }
}
```

Note that this is the same argument structure as the validation example above, but is sent to the action resource §C17.5 rather than the action invoke resource §C18.2 (and without the **x-ro-validate-only** request parameter either).

The server will return a representation of the action with the choices list for the subcategory constrained appropriately.

2.9 Passing arguments to resources

As noted previously, calling a resource using GET with simple arguments §2.8.1 is straight-forward: the arguments are simply passed as key/value pairs. For example:


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

Passing formal arguments §2.8.2 through to resources that accept a PUT or a POST is also easy: the arguments' string representation 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[19] does not guarantee that resources called using GET and DELETE will receive a body[20]. 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, an argument representation such as:

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

can be encoded[21] to:

```
%7B%0A%20%20%22placedBy%22%3A%20%7B%20%0A%20%20%20%20%22value%22%3A
%20%7B%0A%20%20%20%20%20%22ref%22%3A%20%22http%3A%2F%2F~%2Fobjec
ts%2FABC-123%22%2C%0A%20%20%20%20%7D%0A%20%20%7D%0A%7D%0A
```

The same is true if the list or single value representations are used.

2.10 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.

[19] http://www.w3.org/Protocols/rfc2616/rfc2616.html, sec 4.3 and 9.7.

[20] Emperical testing confirms that bodies are not preserved by servlet containers such as Tomcat and Jetty. Proxies may also strip out the body.

[21] eg, using http://meyerweb.com/eric/tools/dencoder/

2.11 URL encoding and Case sensitivity

The URLs defined by the Restful Objects spec follow the rules defined by the HTTP spec[22]. In particular, this means that URL matching is case sensitive[23], and that certain characters (such as "/", "|", "&", ":") may not be used directly and so 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.

2.12 Caching (Cache-Control)

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, using the standard **Cache-Control** header. This takes the values:

- ***no-cache***
  to indicate no caching, suitable for transactional resources such as domain objects and domain object members;
- ***max-age: nnn***
  to indicate the number of seconds to cache the representation. For example, 3600 (1 hour) is suitable for user resources, while 86400 (1 day) is suitable for domain model resources. A server may also return a max-age for domain object representations that are immutable (such as reference data).

[22] http://www.ietf.org/rfc/rfc1738.txt;

[23] excluding the hostname part of a URL, which is case insensitive.

2.13 Security

2.13.1 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*[24]) 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[25].

2.13.2 Authorisation ("disabledReason")

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

First, if the credentials are such that the object member is hidden/invisible, 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 can, if it wishes, use this information in its user interface (for example as a 'tooltip').

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.

2.14 Concurrency Control (If-Match, ETag)

Restful Objects defines an optional capability §B8 for concurrency control, provided through a combination of the **ETag** HTTP response header and the **If-Match** request header.

[24] http://oauth.net

[25] assuming, that is, that **Cache-Control** header is not set to no-cache.

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 the return an error return code).

Restful Objects does not require that **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.

2.15 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 (**400**).

2.16 Internationalisation

The Restful Objects spec supports internationalization as follows:

- json-property keys in representations are never internationalized
- json-property values in representations that would typically be rendered in a UI are internationalised; all others are not. In particular, json-property date-time values are never internationalized (instead are returned in iso date-time string format)
- Internationalized values are with respect to the **Accept-Language** HTTP header.

The spec explicitly indicates which json-property values should be internationalized in their definition. Broadly speaking, those that are

internationized either represent "friendly" names, or descriptions, or are invalidity/disabled reasons.

Implementations indicate whether they support internationalization through the version resource, §B8.2.

3 OPTIONAL CAPABILITIES

While Restful Objects aims to define a consistent standard for RESTful interactions with domain models, realistically not every implementation of Restful Objects will implement each and every feature. For example, pagination or sorting might be easy for one framework to accomplish, but much more difficult for another.

Implementations advertise the capabilities that they support through the "version" resource, §B8. For example, if pagination is not supported by a particular implementation then the client can determine this and then manage pagination itself.

In some cases, clients can vary the behaviour of these capabilities by providing optional query parameters. For example, if server-side sorting is is supported by the implementatino, then the client can use the "**x-ro-sort-by**" query parameter to specify a certain order of results.

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

If a framework has not implemented some aspect of the Restful Objects specification and can reasonably continue, then it should ignore the request (eg the reserved **x-ro-page-size** query parameter). 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 §2.10.

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
- 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".

Implementations that initially support simple scheme and later add in support for the formal scheme shouldensure backwards compatibility with existing clients by returning "selectable" rather than "formal". IIn the absence of the **x-ro-domain-model** parameter the implementation will continue to return the simple scheme information, and may optionally return the formal scheme information also.

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
domainType	string	domain object	Fully qualified class name
friendlyName	string	domain object, property, collection, action, action param	Version of the name suitable for use in a UI (eg as a label). The value should be internationalized, §2.16.
pluralName	string	domain object	Pluralized form of the friendly name, for use in a UI (eg as a label of a collection). The value should be internationalized, §2.16. The implementation should generate a value if need be.

JSON-Property	Values	Applies to	Description
pluralForm	string	collection action returning list	Pluralized form of the element type within the collection/list. The value should be internationalized, §2.16.
description	string	domain object, property, collection, action, action param	Description, suitable for use in a UI (eg as a tooltip). The value should be internationalized, §2.16.
isService	Boolean	domain object	whether this domain object is a domain service
optional	boolean	property, action param.	if false, then a value for the property / param must be provided. Default is implementation-specific.
format	string	property with string value, action param with string value	indicates the format §2.5 of the string.
maxLength	int	string property, string action param	the maximum number of characters that the string may contain. A value of 0 means unlimited.
pattern	string	string property, string action param	whether value matches the regular expression that any submitted value must match.
returnType	string	property non-void action returning scalar or object action param	If scalar value returned, its datatype §2.5. If object returned, its fully qualified type name.
returnType	string	collection action returning collection	Either 'list' or set'.
elementType	string	collection action returning collection	Fully qualified type name of the elements held within the collection.
hasParams	boolean	action	whether an action has parameters or not. This may, for example, be used by clients to render ellipsis (…) in their UI.
memberOrder	int	property, collection, action	a presentation hint as to the relative order to display each member

Implementations are free to extend this list as they require.

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 these resources are obtained by **rel**="**domainModel**" 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 property int id {get; set; }
  public property date since {get; set; }
  public boolean 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) { ... }
  ...
}
```

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

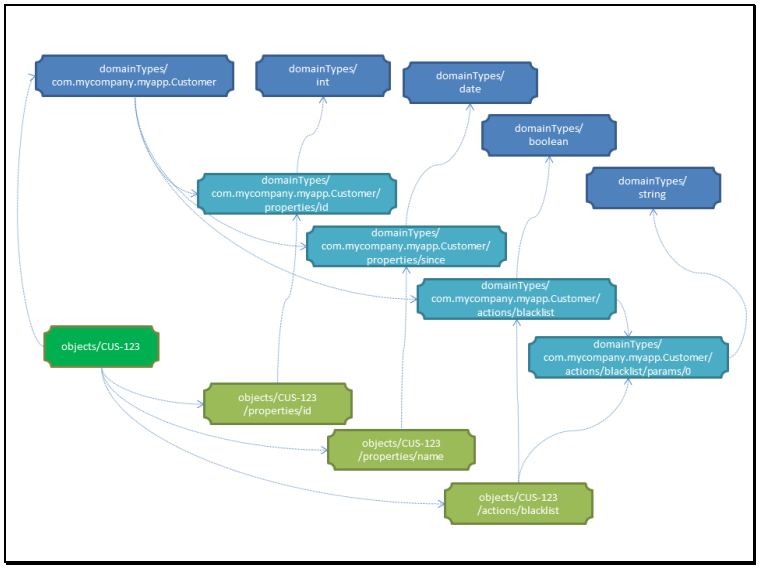

FIGURE 2: DOMAIN OBJECTS VS DOMAIN TYPES

If the type being represented is a scalar type §2.5 then that is used as the domain type name. Otherwise the fully qualified type name is used.

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

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

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

3.2 Minimizing Round-trips (x-ro-follow-links)

While HTTP caching §2.12 works well enough for non-transactional resources, most of the resources served up by Restful Objects will be transactional. Therefore, Restful Objects also defines an optional capability § B8 to support 'eager following' of links. This is specified by setting the reserved **x-ro-follow-links** query parameter. This acts as a hint to the server to generate in its response a representation that includes additional information as a result of following links.

For example, the client can 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 is typically 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 the each member of the object:

```
"members": {
  "createdOn": {
    "memberType": "property",
    "value": ...,
    "links": [ {
      "rel": "details",
      "href": "...",
      ...
    }, ... ]
  },
  "customer": {
    "memberType": "property",
    "value": ...,
    "links": [ {
      "rel": "details",
      "href": "...",
      ...
    }, ...]
  },
  "items": {
    "memberType": "collection",
    "links": [ {
      "rel": "details",
      "href": "...",
      ...
    }, ...]
  },
  "confirm": {
    "memberType": "action",
    "links": [ {
      "rel": "details",
      "href": "...",
      ...
    }, ...]
    ...
  }
]
```

A common use of **x-ro-follow-links** is to request the population of a "value" json-property for any node in the map. For example:

- members.items

  will populate the "value" json-property of the items collection.

- members[memberType=property].links[rel=details]

  follows the "details" link of every object property

- members.confirm.links[rel=details]

  follows the details link of the confirm() action

In all these cases the identified elements are links; the returned representation will include a "value" json-property for the identified links

As an alternative to using paths, the **x-ro-follow-links** may specify a well-defined ("precanned") value that is defined by that resource. For example, the GET Object resource §C14.1 defines "ObjectEdit" as a hint to additionally include property details.

Note however that the support of **x-ro-follow-links** query parameter is an optional capability §B8. If a particular server does not support this capability, then the client must be able to follow the links itself.

If the parameter is present and contains a value that does not represent a link or is otherwise not understood by the server, then the server will silently ignore the query parameter.

3.3 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 **400** "bad request" 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 **400** "precondition failed" with corresponding **Warning** header / "**invalidReason**" json-properties.

3.4 Sorting (x-ro-sort-by / "sortedBy")

Restful Objects defines an optional capability §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 the reserved **x-ro-sort-by** query param. If present, this parameter specifies a comma separated list of sort properties, indicating ascending or descending for each (similar to an ORDER BY statement in SQL).

For example:

- mostRecentOrder.placedOn desc, lastName, firstName

for a resource returning a list of links to Customers 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 are 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 includes the "**sortedBy**" json-property. For example:

```
{
  "sortedBy": [{
      "clause": "mostRecentOrder.placedOn",
      "direction": "desc"
    }, {
      "clause": "lastName",
      "direction": "asc"
    }, {
      "clause": "firstName",
      "direction": "asc"
    },
    ...
  ],
  "value": [
    ...
  }
```

Note that the "**sortedBy**" json-property is a list (rather than a map) because the order of keys in a JSON map is not guaranteed.

3.5 Pagination (x-ro-page, x-ro-page-size, "pagination")

Restful Objects defines an optional capability §B8 to allow object lists §B11 (as returned from action invocations) to be paginated.

If supported, the client may optionally request that a returned list be paginated, by setting the reserved **x-ro-page** query parameter to specify which page of objects is being requested, and the **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 includes a "**pagination**" json-property, which in turn defines "**page**", "**pageSize**" and "**numPages**" json-propertys, as well as the "**totalCount**". In addition, it provides a "**links**" json-property that lists the links to the previous and next pages.

For example:

```
{
  ...
  "pagination": {
    "page": 3,
    "numPages": 4,
    "pageSize": 25,
    "totalCount": 82,
    "links": [ {
        "rel": "previous",
        "href": ...,
        "type": ...,
      }, {
        "rel": "next",
        "href": ...,
        "type": ...,
      }
    ]
  }
  "value": [
     ...
  ]
}
```

Using this information the client can manage the paging, for example enabling/disabling *next* and *previous* buttons in its UI.

3.6 Attachments (x-ro-attachments)

As well as properties representing strings and dates, etc, the specification also supports properties whose value is a blob or a clob §2.5. A typical example is a property representing a media item such as a picture or document.

Under normal circumstances, the value of these properties is suppressed from the object representation §C14.4, §**Error! Reference source not found.**, but is inlined within the property representation §C16.4. The values of blob or clob properties are set/cleared using PUT (§C16.2) and DELETE (§C16.3), as for any other property.

Attachments refer to the optional capability of implementations to also suppress the inline value of the blob/clob within the property representation, and instead to provide a "**rel**"="attachment" link . If followed, such a link returns a representation with the appropriate content-type, eg 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",
    "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.

If attachments are supported then the PUT resource for the property representation C16.2 may also accept also allows the body to provide the value in its native media type (eg image/jpeg).

The client can determine the behaviour of the server with regard to attachments by checking the *version* resource §B8. It can also override the implementation's default behaviour using the **x-ro-attachments** request parameter:

"attachments" capability	x-ro-attachments query parameter	Description
enabled	--	Attachment capability enabled so that attachment links provided, value of blob/clob suppressed from property representation §C16.
enabled	"disable"	Disables attachment capability so that attachment link not provided, value of blob/clob are inlined.
available	"enable"	Enables attachment capability so that attachment links provided, value of blob/clob suppressed.
available	--	Does not enable attachment capability, ie attachment links not provided, value of blob/clob inlined within property representation §C16.
none	(does not matter)	Attachment link not provided, value of blob/clob inlined.

The values of blob or clob properties are set/cleared using PUT (§C16.2) and DELETE (§C16.3), as for any other property. However, the PUT resource also allows the body to provide the value in its native media type (eg image/jpeg).

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 with fixed meanings, irrespective of which representation they appear within. They are:

JSON-Property	Description
disabledReason	provides the reason (or the literal "disabled") as to 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).
invalidReason	provides the reason (or the literal "invalid') as to why a proposed value for a property, collection or action argument is invalid. Appears within an argument representation §2.8 returned as a response.
x-ro-invalidReason	provides the reason as to why a **SET OF** proposed values for a properties or arguments is invalid. The "x-ro-" prefix is to avoid name clashes with the property/argument names.
pagination	indicates that a paginated subset of links in the list representations §B11 has been returned.
sorting	indicates that the order in which values of links in a list representation §B11 or collection property representation §C17.5 are returned.
links	a list of additional links from the representation to other resources. Implementation-specific links may also be present in this list; see §2.10.
extensions	map of additional information about the resource. Implementation-specific json-properties may also be present in this map; see §2.10.

4.2 Specified (reserved) query parameters

The query parameters reserved by Restful Objects are:

Header	Description
x-ro-follow-links	Semi-colon-separated list of either paths of links to follow and include in response, or of well-defined keywords that imply the same (eg "ObjectEdit").

Header	Description
x- ro-page	Requesting page number for a resource that returns a list representation. Typically called with X-Page-Size, if latter is omitted, then a default page size (defined by the implementation) is used.
x- ro-page-size	Specifying pages of a certain size for a resource that returns a list representation. Typically called with X-Page; has no effect if X-Page is not also passed.
x-ro-sort-by	Sort order of links for a resource that returns a list representation, or collection property representation §C17.5 , as a comma-separated list of clauses.
x-ro-domain-model	Which domain model scheme §2.5 the server should return. Understood only by servers that support both domain model schemes (provide "selectable" for the "domainModel" capability §B8)
x-ro-validate-only	Indicates that parameters should be validated but no change in state should occur.
x-ro-attachments	Indicates whether attachment capability §A-48, if available, should be enabled or disabled.

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 **ETag** 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")

Header	Description
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 representation types or 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 (eg 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 (eg If-Unmodified-Since header), or a warning message generated by the application.
401	Not authorized	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 **Accept** 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.

Code	Description	When
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. 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.

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[26]

- **x-ro-follow-links** (optional)
  - user
    - eagerly load currently logged-in user
  - services.value
    - eagerly load domain services representations

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"

[26] http://en.wikipedia.org/wiki/Query_string

- **Cache-Control**:
  - max-age: 86400 (1 day)
    - since home page changes only on redeployment

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:

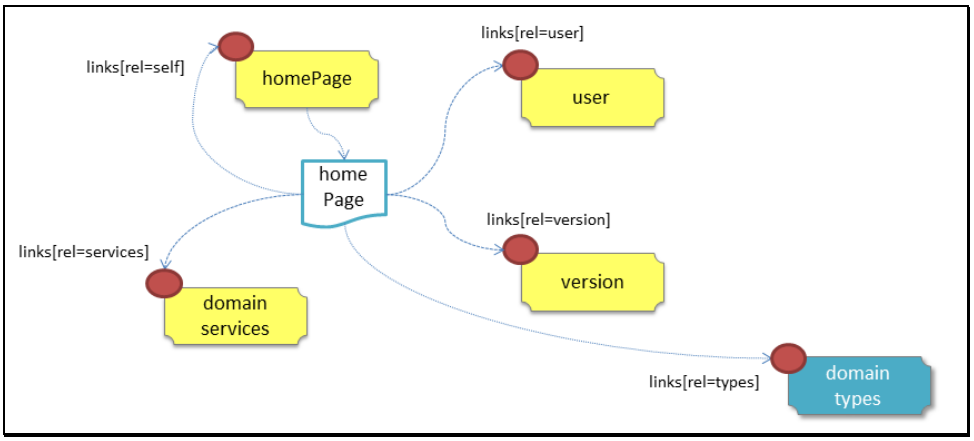

FIGURE 3: HOME PAGE REPRESENTATION

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

where:

JSON-Property	Description
links	list of links to resources.
link[rel=self]	link back to the resource that generated this representation.
link[rel=user]	link to the user resource §6
link[rel=services]	link to the services resource §7
link[rel=version]	link to the version resource §8
link[rel=types]	link to the domain types resource §D20
extensions	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. For further information on security concepts, see §A2.13.

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

- **x-ro-follow-links** (optional)
  - 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"
- **Cache-Control**
  - max-age: 3600 (1 hour)

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:

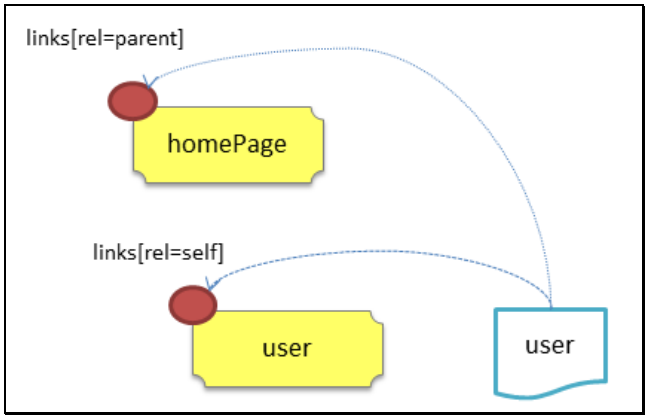

FIGURE 4: USER REPRESENTATION

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 long form, eg so that it can be rendered in a UI.

JSON-Property	Description
email	(optional) the user's email address, if known
roles	list of unique role names that apply to this user (may be empty).
extensions	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

- **x-ro-follow-links** (optional)
  - value
    - eagerly load representations for all domain services

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 "
- **Cache-Control**:
  - max-age: 86400 (1 day)

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

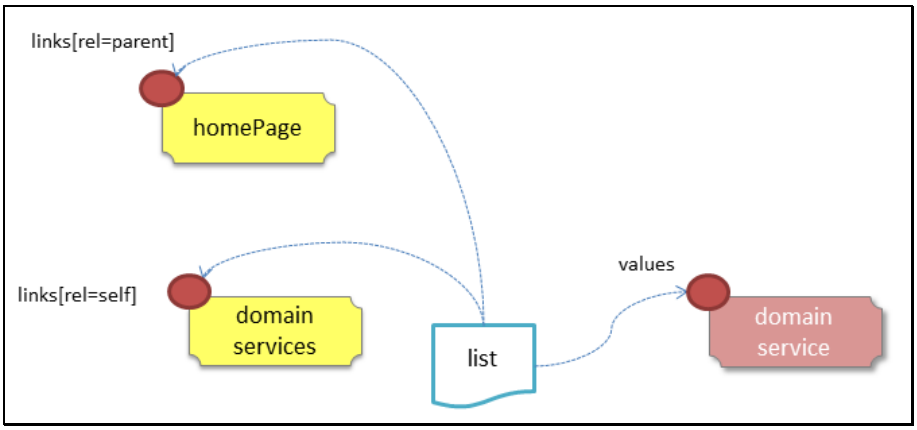

FIGURE 5: SERVICES REPRESENTATION

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, and which of the optional capabilities 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"

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:

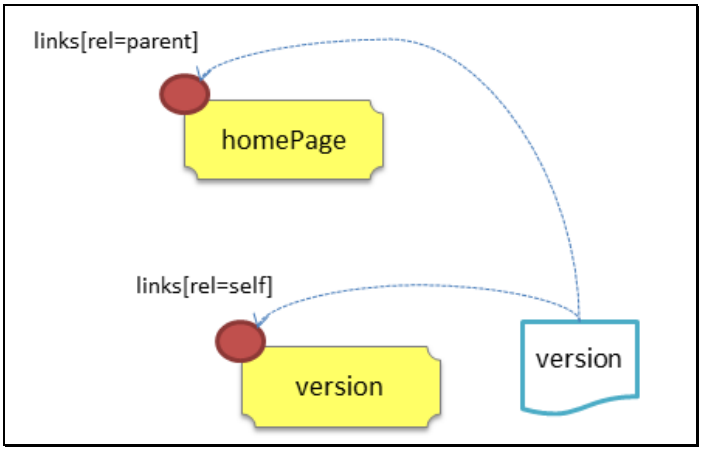

FIGURE 6: VERSION REPRESENTATION

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": "0.61",
  "implVersion": …,
  "optionalCapabilities": {
    "concurrencyChecking": "yes",
    "transientObjects", "yes",
    "deleteObjects", "no",
    "partialArguments", "no",
    "followLinks", "yes",
    "validateOnly", "no",
    "sorting", "no",
    "pagination", "no",
    "domainModel": "formal",
    "attachments": "available",
    "i18n": no
  },
  "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 home page resource, §5.
specVersion	Version of the Restful Objects spec supported by this implementation
implVersion	Version of the implementation itself.
optionalCapabilities	list of strings representing the capabilities supported by this implementation (see below)
extensions	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	Meaning
concurrencyChecking	yes no	Whether concurrency checking using the **If-Modified-Since** and **Last-Modified** headers §A2.14 is supported.
transientObjects	yes no	Whether special handling of transient objects is supported
deleteObjects	yes no	Whether the DELETE Object resource §C14.3 (to delete persisted objects) is supported
partialArguments	yes no	Whether action representations §C18.2 reflecting dependencies between parameters can be generated from a partial argument map to action resource §C17.5.
followLinks	yes no	Whether the **x-ro-follow-Links** reserved query parameter §A3.2 is supported.
validateOnly	yes no	Whether the reserved **x-ro-validate-only** §A3.3 query parameter is supported
sorting	yes no	Whether sorting using the reserved **x-ro-sort-by** query parameter §A3.4 is supported
pagination	yes no	Whether pagination using the reserved **x-ro-page** and **x-ro-page-size** query parameters §A3.5 is supported.

Capability	Value	Meaning
domainModel	none simple formal selectable	The scheme by which the domain model is represented §A2.5. A value of "selectable" means that the reserved **x-domain-model** query parameter is supported.
attachments	enabled available none	Whether attachments §A3.6 are supported.
i18n	yes no	Whether internationalization of values (honouring the **Accept-Language** header) is supported or not.

"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 RESOURCE

The Objects (optional) resource is the target of a persist link to persist a transient object § A2.2. It accepts a representation of a transient domain object and then persists it.

The overall process is:

- client invokes an action that creates the transient object representation
- client uses the "**arguments**" json-property of the rel=persist link to determine the information required
- client provides this information
- client posts the arguments map back to the Objects resource
- assuming that the values are valid, a representation of the newly persisted domain object is returned. This will now include the "self" link and "**oid**" json-property to identify it.

The endpoint for this resource is:

```
/objects
```

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. [27] It consists of:

- members[memberType=property]
- domainType
  - if the simple scheme for domain model information is used

[27] Note that this is different from the body provided to PUT Object §C14.2 used to update multiple properties.

- links[rel=describedby]
  - if the formal scheme for domain model information is used

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

In addition, it may include the reserved query parameter:

- **x-ro-validate-Only**
  - "true"
    - only validate the request, do not persist a new object

For example:

```
{
  "links": [ {
    "rel": "describedby",
    "href": "http://~/domainTypes/x.Customer"
  }],
  "members": {
    "firstName": {
      "value": ...
    },
    "lastName": {
      "value": ...
    },
    ...
  ]
}
```

9.1.2 POST Successful Response

As per §C13.1 (200), 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

9.1.4 POST Validation Failed Response

9.1.4.1 Status Code

- 400 "Validation failed"

9.1.4.2 Headers

- **Warning**
  - summary message

9.1.4.3 Body

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

For example:

```
{
  "links": [ {
    "rel": "describedby",
    "href": "http://~/domainTypes/x.Customer"
  }],
  "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. This may or may not be internationalized §A2.16, dependent on the the implementation.
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 either to domain object resources. Typically it is obtained as the result of invoking a query action, when it is inlined within the action result representation §C19.4. However, it is also be obtained directly from the Services resource §B7, when it provides a list of links to domain service resources.

The representation consists of the following json-properties:

```
{
  "value": [
    ...
  ],
  "links": [ {
      "rel": "self",
      ...
    }, {
      "rel": "elementtype",
      ...
    },
  ],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
links	list of links to resources.
links[rel=self]	(optional) link to the resource that generated this list. Present only for domain services.
links[rel=elementtype]	link to the domain type for the elements within the list – if the "domainModel" optional capability §B8 is implemented.
value	the actual list of links to the domain object resources (or domain service resources)
extensions	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 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.

Lists of (links to) domain services

The same representation is also used to return a list of domain services (because domain services are just well-known domain objects). The only difference is that the "**href**" will be a link to a domain service resource §C15 rather than a domain object resource §C14.

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 inlined within the representation of an action invocation §C19.4 if the action returned a scalar type such as a string or an integer.

The representation consists of the following json-properties:

```
{
  "value": ...,
  "links": [ {
      "rel": "returntype",
      ...
    },
    ...
  ],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
links	list of links to other resources.
links[rel=returntype]	(optional) link to the domain type of the value – if the "formal" domain capability §B8 is supported.
value	the scalar value itself. If text is returned then this should be internationalized, §A2.16.
extensions	additional metadata about the representation.

Note that NO "**self**" link is present, because the scalar value is not a persisted resource (is transient).

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
- 204 – Request succeeded, but generated no representation; or validation succeeded
- 400 – Either syntactically invalid request, or validation failed
- 401 – Not authorised (user not authorised to access 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 – Precondition failed (object changed by another user)
- 500 – Domain logic failed

The following table indicates which of these status return codes may be generated for each resource:

	Method	Res.	Repr.	200	204	400	401	404	405	406	412	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
	DELETE	§C14.3	---		y	y	y	y	y	y	y	y
Property	GET	§C16.1	§16.4	y		y		y		y		y
	PUT	§C16.2	---	y		y	y	y		y	y	y
	DELETE	§C16.3	---	y		y	y	y		y	y	y
Collection	GET	§C17.1	§17.5	y		y		y		y		y
	PUT	§C17.2	---	y		y	y	y	y	y	y	y
	POST	§C17.3	---	y		y	y	y	y	y	y	y
	DELETE	§C17.4	---	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
	PUT	§C19.2	---	y		y	y	y	y	y	y	y
	POST	§C19.3	---	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 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 (eg modifying a property or collection or invoking an action), then there will be no self link. This prevents clients from easily bookmarking the link.

13.1.1 Status code

- 200 "OK"

13.1.2 Headers

- **Content-Type:**
  - application/json;profile=".../xxx;x-ro-domainType=yyy"
    - where xxx indicates the representation type
    - where yyy is the fully qualified type name
- **Cache-Control**
  - no-cache
    - if the object is transactional
  - max-age: nn
    - if the implementation can determine that the returned representation is safe to cache (eg 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, but generated no content

This response is most often generated as the result of a validation succeeding (if **x-ro-validate-only** is supported, §A3.3). Note however that invoking a void action DOES return a representation §19.4.4.

13.2.1 Status code

- 204 "No content"

13.2.2 Headers

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

13.2.3 Body

- empty

13.3 Bad request

May be generated either as the result of a syntactically invalid request, or as the result of validation failure.

13.3.1 Status code

- 400 ("bad request")
  - missing arguments
  - arguments are malformed
  - property member values are invalid (if updating multiple properties §14.2, or if persisting a transient object §B9.1)
  - mandatory header missing (eg **If-Match**)
  - "Arguments invalid"
    - details are provided in the body

13.3.2 Headers

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

13.3.3 Body

If arguments §A2.8.2/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 (eg 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

13.4 Forbidden (user not authorised to access resource)

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

13.4.1 Status Code

- 401 "Not authorised"

13.4.2 Headers

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

13.4.3 Body

- empty

13.5 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.

13.5.1 Status Code

- 404 "Not found"

13.5.2 Headers

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

13.5.3 Body

- empty

13.6 Resource has invalid semantics for method called

13.6.1 Status code

- 405 ("method not allowed")

13.6.2 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)

13.6.3 Body

- empty

13.7 Not acceptable

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

13.7.1 Status code

- 406 ("not acceptable")

13.7.2 Headers

- none

13.7.3 Body

- empty

13.8 Precondition failed (object changed by other user)

13.8.1 Status code

- 412 "precondition failed"

13.8.2 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

13.8.3 Body

- empty

13.9 Domain logic failed, or Implementation defect

13.9.1 Status code

- 500 ("internal server error")

13.9.2 Headers

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

13.9.3 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 instance, and can also be used to update or delete individual persisted domain object instances.

The endpoint URL for this resource is:

```
/objects/{oid}
```

where:

- {oid} is the object identifier

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. If required, the reserved **x-ro-follow-links** query parameter can be used to request additional information in the generated representation.

14.1.1 Request

14.1.1.1 Query String

- **x-ro-follow-links** (optional)
  - members.domainModel
    - eagerly load domain model metadata about all members
  - members.propertyDetails
    - eagerly load details about all properties
  - members.collectionDetails
    - eagerly load details (including their contents) of all collections
  - members.collectionContents
    - eagerly load the items for all collections
  - embers.collectionCount
    - eagerly load the count for all collections
  - members.actionDetails
    - eagerly load details about all actions
  - ObjectEdit
    - equivalent to specifying members.domainModel;members.propertyDetails

The **x-ro-follow-links** request param can also be targeted to specific members by using an xpath qualifier on members; for example:

```
members.items.collectionContents;members.overdue.collectionCount
```

14.1.1.2 Headers

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

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 property, or to request validation of the proposed value using the **x-ro-validate-only** query param §A3.3.

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 §A2.8.2.3. Note that any blob/clob properties must be inlined 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 parameter:

- **x-ro-validate-only** (optional)
  - "true"
    - only validate the request, do not modify the property

For example:

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

14.2.2 Success Response

As per 13.2 (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 §13.2 (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 the single most important representation defined by Restful Objects.

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

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

where yyy is the fully qualified type name of the returned object.

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 an action invocation returning a single domain object (§18.2), or as the result updating multiple properties §14.2, or of persisting a transient object §B9.

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

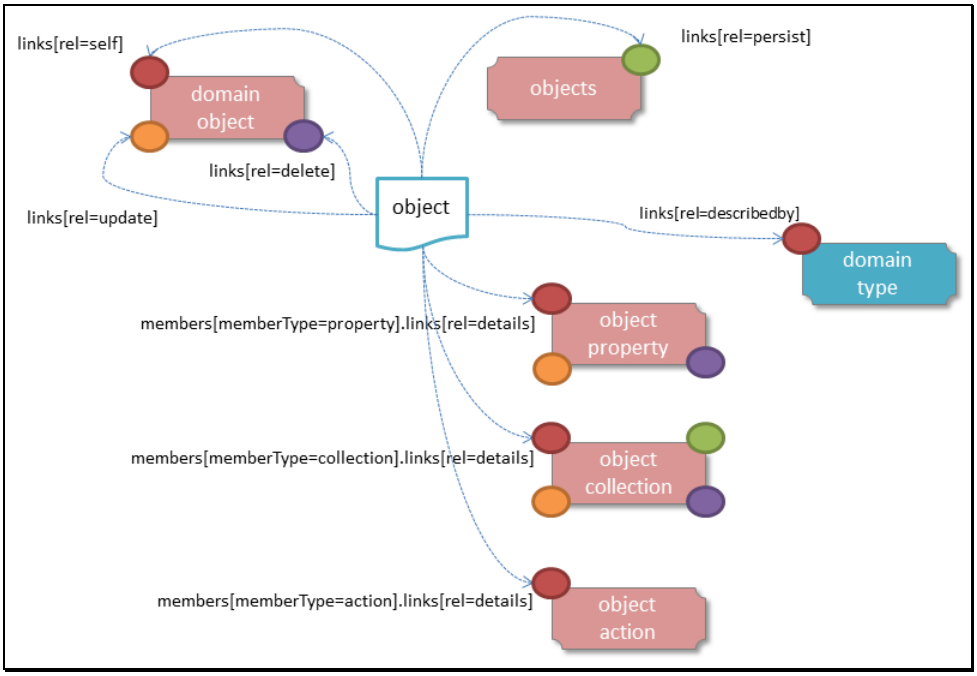

FIGURE 7: DOMAIN OBJECT REPRESENTATION

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

```
{
  "oid": "ORD-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
links	list of links to other resources.
links[rel=self]	(optional); link to a resource that can obtain this representation. Note that the href for a service will be http://~/services/{serviceName}. Discussed further below.
links[rel=modify]	(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.
oid	(optional) the object identifier, to use when building template URIs. Discussed further below.
serviceId	(optional) the service Id. Present only if the object is a domain service §15.
title	a string identifier of the object, suitable for rendering in a UI.
members	map of object members (properties, collections, actions)
links[rel=persist]	(optional) persist the (transient) domain object. Discussed further below.
links[rel=update]	(optional) update properties of the (persistent) domain object. Discussed further below.
links[rel=delete]	(optional) delete the (persistent) domain object. Discussed further below.
links[rel=iicon]	(optional) link to an image representing a scalable icon for this object
links[rel=icon32]	(optional) link to an image to represent this icon at 32x32 pixels
links[rel=icon16]	(optional) link to an image to represent this icon at 16x16 pixels
extensions	additional information about the resource.

"oid" and "links[rel=self]"

The "**oid**" json-property is present for persistent domain entities and for addressable view models §A2.2, and can be used to construct URLs to other resources for the domain obect as required.

Transient domain entities and (non-addressable) view models do not have an "**oid**" because they do not correspond to any server-side state that can be directly addressed.

For the same reasons, the link to self will only be present whenever an "**oid**" json-property is present.

"members"

The "**members**" map contains an entry for every (visible) member. It is described in more details in the sections below §14.4.1, §14.4.2, §14.4.3[28].

"links[rel=update]" and "links[rel=delete]"

For persistent domain objects, these may optionally have a **rel="update"** link to update all properties and a **rel="delete"** link to delete the domain object.

These links are 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, and if the object may not be deleted then the delete link will not be present either.

Transient domain objects and view models will never have these links.

"links[rel=persist]"

For transient domain objects that can be persisted, 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**").

[28] The reserved **x-ro-follow-links** query parameter may also be used to request that more detailed information is returned in the representation.

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://~/domaintypes/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

This 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",
      "href": "http://~/objects/ORD-123/properties/createdOn",
      "type": "application/json;profile=\".../objectproperty\"",
      "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
memberType	the constant value "property"
value	(optional) the current value of the property, either a scalar, a (link representing a) reference or null. Discussed further below.
disabledReason	(optional) if populated then indicates the reason why the property cannot be modified.
links	list of links to resources.
links[rel=details]	(optional) link to the detailed representation of the property, §16.4 (eg to access defaults and choices).
links[rel=attachment]	(optional) link to the property value if an attachment. Discussed further below
extensions	map of additional information about the resource.

"value" and "links[rel=attachment]"

The "value" json-property holds the inlined 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 transient domain objects and (non-addressable) view models (§A2.2), the "**value**" is always present.
- for persistent domain objects and actionable 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-48, again the "**value**" is present
- however, the "**value**" is omitted for persistent domain objects which support attachments. Instead a link to its attachment "**links[rel=attachment]**" will be available. This link serves up the property value directly with the correct media type (eg as an image/jpg).

From the client's perspective, this boils down to there either being 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 that may be available 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.

Typically, this entry provides 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 (i§A2.2), then the collection's representation also inlines the value of the domain object 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",
      "href": "http://~/objects/ORD-123/collections/items",
      "type": "application/json;profile=\".../objectcollection\"",
      "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
memberType	the constant value "collection"
disabledReason	(optional) if populated then indicates the reason why the collection cannot be modified.
value	(optional) contains the list of the links to elements within the collection. Discussed further below.
size	(optional) contains a count of the elements in the collection. Discussed further below.
links	links to other resources.
links[rel=details]	(optional) link to the detailed representation of the collection, §17.5, which includes such information as defaults and choices. Discussed further below.
extensions	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 inline 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).

However, the **x-ro-follow-links** query parameter (if supported §A3.2) can be used to influence this behaviour:

- setting the query parameter to "**links[rel=details]**" will 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**" will cause the optional "**value**" to be returned, holding a list of links to the actual elements. These links will have their "**title**" json-property §A4.1 populated;
- setting the query parameter to "size" will 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 may be available either in the "**value**" json-property, or may be in the inlined details representation "**links[rel=details].value**" json-property.

"links" and "extensions"

Other domain model information about the collection is available through either the "**links**" or the "**extensions**" json-properties. The information that may be available 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 a value 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 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 properties and collections, actions have a link to 'details' which allows additional information (specifically, choices and defaults on parameters) to be obtained that might otherwise be expensive to compute. They also indicate the link to follow in order to invoke the action.

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

```
"members": {
  ...
  "submit": {
    "memberType": "action",
    "disabledReason": ...,
    "links": [ {
      "rel": "details",
      "href": "http://~/objects/ORD-101/actions/submit",
      "type": "application/json;profile=\".../objectaction\"",
      "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
memberType	the constant value "action"
disabledReason	(optional) if populated then indicates the reason why the action may not be invoked.
links	list of links to other resources.
links[rel=details]	link to the detailed representation of the action, §18.2.
extensions	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 that may be available 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": "Orders that have been placed by customers",
  "isService": false
}
```

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://~/domainTypes/x.Order",
    "type": "application/json;profile=\".../domaintype\"",
    "method": "GET"
  },
  ...
]
```

which links to the domain type resource §D21 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.

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. [29].

The endpoint URL for this resource is:

```
/services/{serviceName}
```

where:

- {serviceName} is the fully-qualified class name of the service

15.1.1 Request

15.1.1.1 Query String

- **x-ro-follow-Links** (optional)
  - members.domainModel
    - eagerly load domain metadata about all members
  - members.actionDetails

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.

[29] Since the representation obtained from this resource is just a domain object representation, it is possible to use the {oid} and also access domain services using the GET Domain Object resource §C14. However, this is discouraged.

16 OBJECT 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/{oid}/properties/{propertyId}
```

where:

- {oid} is the object identifier
- {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

16.1.1.1 Query String

- **x-ro-follow-links** (optional)
  - domainModel
    - eagerly load domain model metadata

16.1.1.2 Headers

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

16.1.1.3 Body

- N/A

16.1.2 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 validate that the proposed new value for the property is valid. 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 (any type, application/json)

The first style of request defines the value inlined within a JSON map §A2.8.2.2. This style also allows the client to request validation of the proposed value using the reserved **x-ro-validate-only** query parameter §A3.3.

16.2.1.1 Query String

- none

16.2.1.2 Headers

- **Content-Type:** application/json
- **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 §A2.8.2.2.
- **x-ro-validate-only** (optional)
  - "true"
    - only validate the request, do not modify the property

The above format may also be used for blob/clobs, with the value inlined and URL encoded if required. However, the value for a blob/clob

16.2.2 Request (blobs/clobs)

For properties that are blobs or clobs, their value may also be updated simply by PUTting the new value.

Note that this request style does not allow validation to be requested, not does it allow for optimistic locking checks (**If-Match** checks)

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

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. Because the resource has mutated the state, there will be no self link (so that it cannot be bookmarked by clients).

16.3 HTTP DELETE

This is the recommended resource for clearing property value, or alternatively for validating that a property can be cleared but making no change to the object.

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-validate-only** (optional)
  - "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).

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

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

Typically the representation is obtained from the GET Object Property resource §16.1, but it can also be obtained using the **x-ro-follow-links** qury param when requesting the Get Object resource §14.1.

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

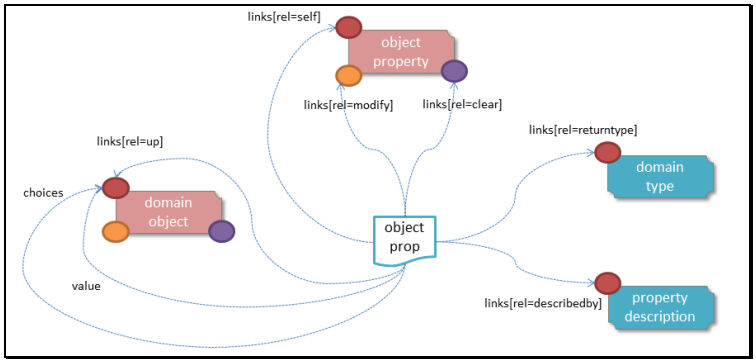

FIGURE 8: OBJECT PROPERTY REPRESENTATION

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

```
"deliveryOptions": {
  "disabledReason": ...,
  "value": ...,
  "choices": [ ... ]
```



```
  "links": [  {
     "rel": "self",
     "href": "http://~/objects/ORD-123/properties/deliveryOptions",
     "type": "application/json;profile=\".../objectproperty\"",
     "method": "GET",
    }, {
      "rel": "modify",
      ...
    }, {
      "rel": "clear",
      ...
    }, {
      "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
value	the current value of the property.
choices	optional list of suggested/recommended choices for the property
disabledReason	(optional) if populated then indicates the reason why the property cannot be modified. The value should be internationalized, §A2.16.
links[rel=modify]	optional link back to self to modify property value; discussed below, §16.4.2.
links[rel=clear]	optional link back to self to clear property value; discussed below, §16.4.2.
links[rel=up]	link to the object that is the owner of this property.
extensions	additional information about the resource.

"choices"

The "**choices**" json-property lists a set of values which are valid for the property. Whether other values might also be valid with respect to the domain model is implementation-specific.

"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 required.

16.4.1 Property values and choices

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

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

For value properties, the "**value**" and "**choices**" json-properties are directly parseable strings:

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

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

where:

JSON-Property	Description
links[rel=modify]	link back to self to modify property value; not included if the property is disabled
links[rel=clear]	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 §A3.3.

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

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

where:

JSON-Property	Description
disabledReason	indicates the reason why the property cannot be modified/cleared; only included if the property is disabled. The value should be internationalized, §A2.16.

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://~/domainTypes/x.Order/properties/deliveryTime",
    "type": "application/json;profile=\".../typeproperty\"",
    "method": "GET"
  }
]
```

which links to the domain property description resource §D21.2 corresponding to this domain object property.

17 OBJECT 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/{oid}/collections/{collectionId}
```

where:

- {oid} is the object identifier
- {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-follow-links** (optional)
  - *timestamp*
  - domainModel
    - eagerly load domain model metadata

17.1.1.2 Headers

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

17.1.1.3 Body

- N/A

17.1.2 Success Response

As per §13.1 (200), returning an object collection representation §17.5.

17.2 HTTP PUT

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 set semantics (the most common case, where duplicate entries are not permitted).

17.2.1 Request

17.2.1.1 Query String

- none

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 §A2.8.2.2.
- **x-ro-validate-only** (optional)
  - "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. Because the resource has mutated the state, there will be no self link (so that it cannot be bookmarked by clients).

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 §A2.8.2.2.
- **x-ro-validate-only** (optional)
  - "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. Because the resource has mutated the state, there will be no self link (so that it cannot be bookmarked by clients).

17.4 HTTP DELETE

Remove an object from a collection, or alternatively validate that an object can be removed from a collection but make no changes.

17.4.1 Request

17.4.1.1 Query String

A single query argument should be formatted as a single argument node §A2.8.2.2 referencing the object to remove:

```
{
  "value": {
    "rel": "object",
    "href": "http://~/objects/XXX-yyyy",
    "method": "GET"
   }
}
```

In addition:

- **x-ro-validate-only** (optional)
  - "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 type, 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="...objectcollection"
```

Typically the representation is obtained from the Object Collection resource §17.1, but it can also be obtained using the **x-ro-follow-links** request parameter when requesting the Object resource §14.1.

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

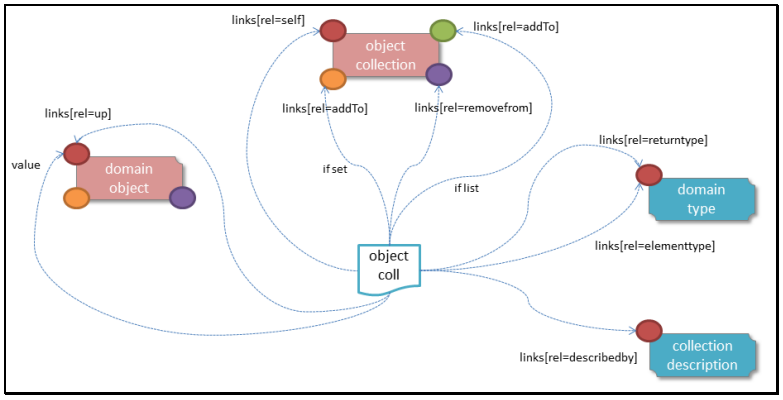

FIGURE 9: OBJECT COLLECTION REPRESENTATION

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

```
{
  "id": items",
  "value": [ ... ],
  "disabledReason": ...,
  "links": [ {
      "rel": "self",
      "href": "http://~/objects/ORD-123/collections/items",
      "type": "application/json;profile=\".../objectcollection\"",
      "method": "GET",
    }, {
      "rel": "addTo",
      ...
    }. {
      "rel": "removeFrom",
      ...
    }, {
      "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.
disabledReason	(optional) if populated then indicates the reason why the collection cannot be modified. The value should be internationalized, §A2.16.
links[rel=addTo]	(optional) link back to self to add item to collection; discussed below, §17.5.2.
links[rel=removeFrom]	(optional) link back to self to remove item from collection; discussed below, §17.5.2.
links[rel=up]	link to the object that is the owner of this collection.
extensions	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 required.

17.5.1 Collection values

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

```
"value": [
  {
    "rel": "object",
    "href": "http://~/objects/ORD-123-1",
    "type": "application/json;profile=\".../object\"",
    "method": "GET",
    "title": "Harry Potter and the Goblet of Fire"
  },
  {
    "rel": "object",
    "href": "http://~/objects/ORD-123-2",
    "type": "application/json;profile=\".../object\"",
    "method": "GET",
    "title": "Rubiks Cube"
  },
  {
    "rel": "object",
    "href": "http://~/objects/ORD-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**" json-properties 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",
      "href": "http://~/objects/ORD-123/collections/items",
      "type": "application/json;profile=\".../objectproperty\"",
      "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",
      "href":"http://~/objects/ORD-123/collections/items",
      "type": "application/json;profile=\".../objectproperty\"",
      "method": "POST"
      "arguments": {
        "value": null
      }
    },
    ...
  ],
  ...
}
```

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

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

To summarize:

JSON-Property	Description
links[rel=addTo]	link back to self to add to collection; not included if the collection is disabled
links[rel=removeFrom]	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") as to why the contents of the collection cannot be modified:

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

where:

JSON-Property	Description
disabledReason	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 (eg "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://~/domainTypes/x.Order/collections/items",
    "type": "application/json;profile=\".../typecollection\"",
    "method": "GET"
  },
  ...
]
```

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

18 OBJECT ACTION RESOURCE & REPRESENTATION

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

```
/objects/{oid}/actions/{actionId}
```

where:

- {oid} is the object identifier
- {actionId} is the action identifier

18.1 HTTP GET

Obtain a detailed representation of an action §18.2.

Restful Objects defines an optional capability §B8 that allows this resource to accept a partial argument map §A2.8.2. 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.

18.1.1 GET Request

18.1.1.1 Query String

If the "partialArguments" optional capability § B8 is supported, then partial argument maps may be provided, for example:

```
{
  "category": {
    "value": {
      "rel": "argument",
      "href": "http://~/objects/CGY-BOOKS",
      "method": "GET"
    }
  }
}
```

In addition:

- **x-ro-follow-links** (optional)

18.1.1.2 Headers

- **Accept**
  - application/json
  - application/json;profile="…/objectaction"

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.

The exact contents (specifically, of the "**choices**" json-property) may vary depending on whether a partial argument map was provided

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="…/objectaction"
```

Typically the representation is obtained from the GET Object Action resource §18.2, but it can also be obtained using the **x-ro-follow-links** request parameter when requesting the Object resource §14.1.

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

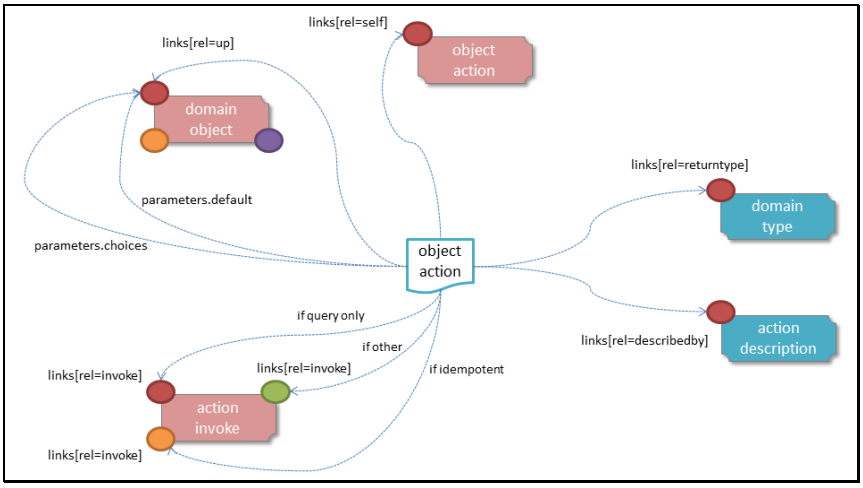

FIGURE 10: OBJECT ACTION REPRESENTATION

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=\".../objectaction\"",
      "method": "GET",
    }, {
      "rel": "invoke",
      ...
    }, {
      "rel": "up",
      ...
    }
    ...
  ],
  "extensions": { ... }
}
```

where:

JSON-Property	Description
links	list of links to other resources.
links[rel=self]	link to a resource that can generate this representation.
id	the action ID, to use when building templated URIs
parameters	map of parameters; discussed below §18.2.1
disabledReason	(optional) if populated then indicates the reason why the collection cannot be modified. The value should be internationalized, §A2.16.
links[rel=invoke]	(optional) is a link to invoke the action (if it is not disabled)
links[rel=up]	link to the object that is the owner of this action.
extensions	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 required.

18.2.1 Action parameters

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

```
"parameters": {
  "paramName1": {
    "choices": [ {
        "rel": "choice",
        ...
      }, {
       "rel": "choice",
        ...
      }, {
       "rel": "choice",
        ...
      }
      ...
     ],
     "default": {
      "rel": "default",
       ...
     },
     "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 A2.8.2), with their value being the following map:

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

In particular, note that 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",
      "href": "http://~/objects/ORD-123/actions/submit/invoke",
      "type": "application/json;profile=\".../actionresult\"",
      "arguments": {
        "shipMethod": {
          "value": "overnight express"
        },
        ...
      },
      "method": "GET"
    }
    ...
  ],
  ...
}
```

"type" property

The "**type**" json-property always indicates that the urn:org.restfulobjects/actionresult 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, as shown for the "shipMethod" parameter in the example above).

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

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

To summarize:

JSON-Property	Description
link[rel=invoke]	link to invoke the action, either as query only, idempotent or neither; 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") as to 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. The value should be internationalized, §A2.16.

18.2.3 Domain model information (for action)

Domain model information is available for action 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://~/domainTypes/x.OrderReceipt",
    "type": "application/json;profile=\".../domaintype\"",
    "method": "GET"
  },
  {
    "rel": "describedby",
    "href": "http://~/domainTypes/x.Order/actions/submit",
    "type": "application/json;profile=\".../typeaction\"",
    "method": "GET"
  },
  ...
}
```

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://~/domainTypes/x.Order/actions/submit/params/product",
        "type": "application/json;profile=\".../typeactionparam\"",
        "method": "GET"
      }
    ]
  }
  ...
}
```


19 OBJECT ACTION INVOKE RESOURCE

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

The endpoint URL for this resource is:

```
/objects/{oid}/actions/{actionId}/invoke
```

where:

- {oid} is the object identifier
- {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 §A3.3.

19.1.1.1 Query String

Query arguments should be formatted as a map (§A2.8.2), and encoded in the URL (§A2.8.2.5). Note that if any argument is a blob/clob, then its value must be inlined (URL encoded for a blob)[30].

[30] 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. 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.

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

- **x-ro-validate-only**
  - the argument map can be incomplete; only those arguments provided will be validated.
- **x-ro-follow-links**
- **x-ro-page**
  - nn; only applies if a list representation (of objects) §B11 is returned
- **x-ro-page-size**
  - nn; only applies if a list representation (of objects) §B11 is returned, and **x-ro-page** query param also provided
- **x-ro-sort-by** (optional)
  - xxx,yyy; only applies if a list representation (of objects) §B11 is returned

19.1.1.2 Headers

- **Accept**
  - application/json
  - application/json;profile="…/actionresult"

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

19.1.1.3 Body

- N/A

19.1.2 Success Response

As per 13.1 (200), returning an action result §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 has the same post-conditions irrespective of whether the order is 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="…/actionresult"
- **If-Match**
    - *timestamp digest*
        - obtained from **ETag** header of representation

19.2.1.3 Body

Arguments should be formatted as a map (§A2.8.2), and sent as the body (§A2.8.2.5). Note that if any argument is a blob/clob, then its value must be inlined (URL encoded for a blob).

In addition:

- **x-ro-validate-only** (optional)
    - "true"
        - only validate the request, do not invoke the action

19.2.2 Success Response

As per 13.1 (200), returning an action result §19.4.

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="…/actionresult"
- **If-Match**
    - *timestamp digest*
        - obtained from **ETag** header of representation

19.3.1.3 Body

Arguments should be formatted as a map (§A2.8.2), and sent as the body (§A2.8.2.5). Note that if any argument is a blob/clob, then its value must be inlined (URL encoded for a blob).

In addition:

- **x-ro-validate-only** (optional)

- "true"
  - only validate the request, do not invoke the action

19.3.2 Success Response

As per 13.1 (200), returning an action result §19.4.

19.4 Representation

First, note that if the "**x-ro-validate-only**" query parameter was passed in and the validation passed, then no representation will be returned. Instead 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 inlines the representation of the result of the invocation.

For example:

```
{
  "links": [ {
       "rel": "self",
       "href":
"http://~/services/TaskRepository/actions/countUrgentTasksFor/invok
e",
       "type": "application/json;profile=\".../actionresult\"",
       "arguments": [
        {
           "rel": "argument",
           "href": "http://~/objects/EMP-090123",
           "type": "application/json;profile=\".../object\"",
           "method": "GET"
        }
     }
  ],
  "resulttype": ...
  "value": ...,
  "extensions": { ... }
}
```

where:

JSON-Property	Description
links	list of links to other resources.
links[rel=self]	(optional) link to the action invocation resource that generated the representation
resulttype	either "object", "list", "scalar" or "void"
result	(optional) the action result itself. Not present if void action.
extensions	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 inlined 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, a list or a 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 inline the domain object's representation (§14.1):

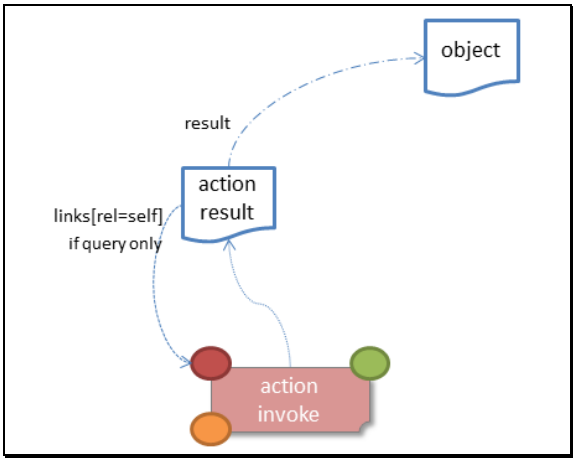

FIGURE 11: ACTION RESULT FOR OBJECT

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=\".../actionresult\"",
      "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 inline a list representation (§B11):

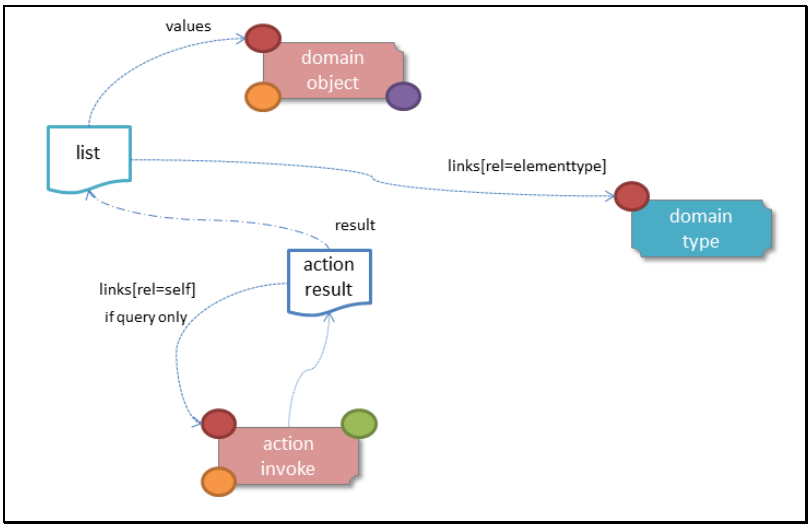

FIGURE 12: ACTION RESULT FOR LIST

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=\".../actionresult\"",
      "arguments": {},
      "method": "GET"
    }
  ],
  "resulttype": "list",
  "result": {
    "links": [{
        "rel": "elementtype",
        "href": "http://~/domainTypes/x.Customer,
        "type": "application/json;profile=\".../domaintype\"",
        "method": "GET"
      },
    ],
    "value": [ {
        "ref": "object",
        "href": "http://~/objects/CUS-123",
        "type": "application/json;profile=\".../object\"",
        "method": "GET"
      }, {
        "ref": "object",
        "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": [ ]
    ...
  }
  ...
}
```

However, it is legal for actions to return a null list. In this case the "**result**" json-property will still be present, but 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 inline a scalar representation (§B12):

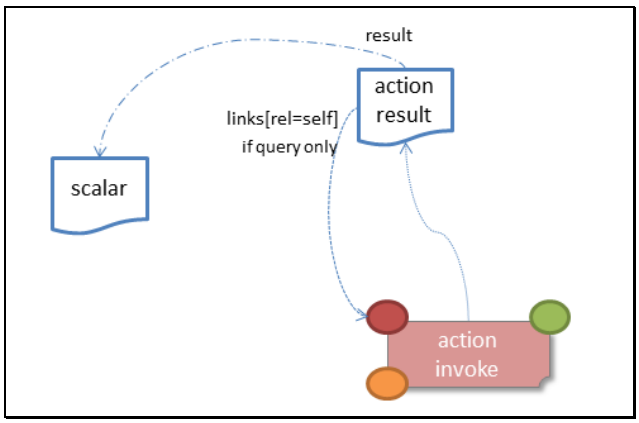

FIGURE 13: ACTION RESULT FOR SCALAR

For example, the TaskRepository's countUrgentTasksFor(Employee) action might generate the following representation:

```
{
  "links": [ {
      "rel": "self",
      "href":
"http://~/services/TaskRepository/actions/countUrgentTasksFor/invok
e",
      "type": "application/json;profile=\".../actionresult\"",
      "arguments": {
         "employee": {
           "rel": "argument",
           "href": "http://~/objects/EMP-090123",
           "type": "application/json;profile=\".../object\"",
           "method": "GET"
         }
      },
      "method": "GET"
    }
  ],
```

```
  "resulttype": "scalar",
  "result": {
    "links": [ {
        "rel": "returntype",
        "href": "http://~/domainTypes/int,
        "type": "application/json;profile=\".../domaintype\"",
        "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 returns a void, then the simple actionresult representation (with no inlined representation) will be returned.

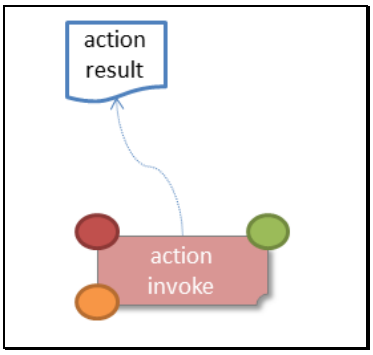

FIGURE 14: ACTION RESULT FOR VOID

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=\".../actionresult\"",
      "arguments": {}
      ],
      "method": "GET"
    },
    ...
  ],
  "resulttype": "void",
  "extensions": { ... }
}
```

Note that there is no "**result**" json-property.

D

DOMAIN TYPE RESOURCES

20 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:

```
/domainTypes/
```

20.1 HTTP GET

20.1.1 Request

20.1.1.1 Query String

- **x-ro-follow-links** (optional)
  - value
    - eagerly load representations for all domain types

20.1.1.2 Headers

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

20.1.1.3 Body

- N/A

20.1.2 Successful Response

20.1.2.1 Status Code

- 200 "OK"

20.1.2.2 Headers

- **Content-Type**
  - application/json;profile=".../typelist"
- **Cache-Control**:
  - max-age: 86400 (1 day)

20.1.2.3 Body

As per §21.2.

20.2 Representation

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

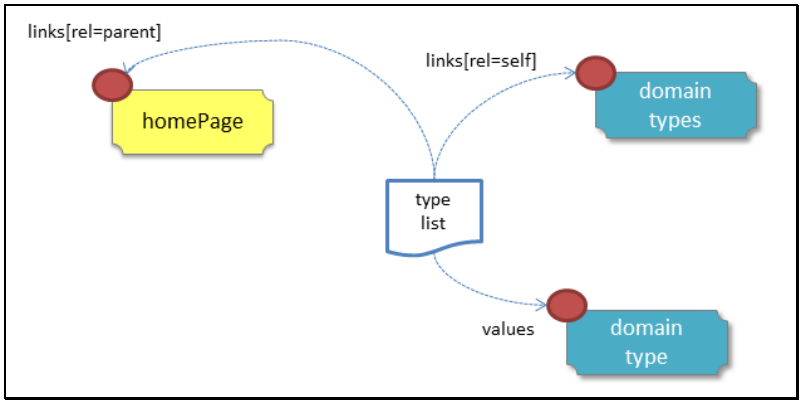

FIGURE 15: DOMAIN TYPE LIST REPRESENTATION

For example, the JSON representation of the collection of domain types for a system looks something like:

```
{
  "links" : [ {
    "rel" : "self",
    "href" : "http://localhost:8080/domainTypes",
    "method" : "GET",
    "type" :
"application/json;profile=\"urn:org.restfulobjects/typelist\""
    }, {
      "rel": "up",
      "href": "http://~/",
      "type": "application/json;profile=\".../homepage\"",
      "method": "GET"
    }
  ],
  "value" : [ {
    "rel" : "domaintype",
    "href" : "http://~/domainTypes/x.Customer",
    "method" : "GET",
    "type" :
  "application/json;profile=\"urn:org.restfulobjects/domaintype\""
  }, {
    "rel" : "domaintype",
    "href" : "http://~/domainTypes/x.Order",
    "method" : "GET",
    "type" :
  "application/json;profile=\"urn:org.restfulobjects/domaintype\""
  },
  ...
  "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, §21
extensions	map of additional information about the resource.

20.3 Predefined Domain Types

There are predefined domain type resources for each of the built-in scalar types §A2.5. Specifically this means that the following URLs are always valid:

- http://~/domainTypes/number
- http://~/domainTypes/string
- http://~/domainTypes/boolean
- http://~/domainTypes/integer
- http://~/domainTypes/date-time
- http://~/domainTypes/date
- http://~/domainTypes/time
- http://~/domainTypes/utc-millisec
- http://~/domainTypes/biginteger
- http://~/domainTypes/bigdecimal(n)
    - where n can be any positive integer
- http://~/domainTypes/blob
- http://~/domainTypes/clob

In addition, there are predefined domain type resources to represent lists and sets:

- http://~/domainTypes/list
- http://~/domainTypes/set

These may only be referenced by collections or by actions that return collections. Having these additional domain type resources enables the format of the "**returnType**" json-property to be consistent across all member types (properties, collections and actions).

No representations are returned

No representations are defined for any of the above resources; instead, a 204 (no content) will be returned. Clients are expected to have built-in support for these domain types (eg a calendar widget to render dates; a checkbox widget to render Booleans,).

21 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:

```
/domainTypes/{domainType}
```

where:

- {domainType} is either
  - the fully qualified type name, or
  - is a built-in JSON type

21.1 HTTP GET

Obtain a representation of a domain type within the metamodel.

21.1.1 Request

21.1.1.1 Query String

- **x-ro-follow-links** (optional)
  - propertyDetails
    - additionally include property details
  - collectionDetails
    - additionally include collection details
  - actionDetails
    - additionally include action details

21.1.1.2 Headers

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

21.1.1.3 Body

- N/A

21.1.2 Successful Response

21.1.2.1 Status Code

- 200 "OK"

21.1.2.2 Headers

- **Content-Type**
  - application/json;profile=".../domaintype"
- **Cache-Control**:
  - max-age: 86400 (1 day)

21.1.2.3 Body

As per §21.2.

21.1.3 Not found Response

21.1.3.1 Status Code

- 404 "Not found"

21.1.3.2 Headers

- **Warning**:
  - No such domain type {domainType}.

21.1.3.3 Body

- empty

21.2 Representation

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

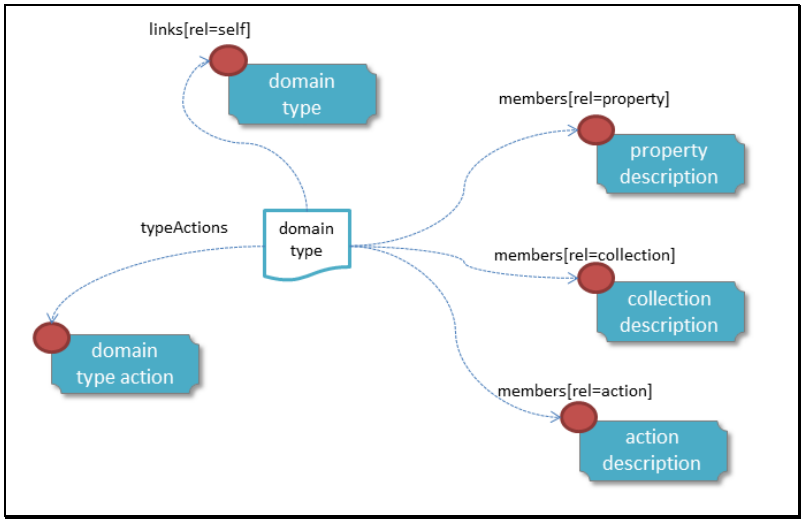

FIGURE 16: DOMAIN TYPE REPRESENTATION

For example, the JSON representation (for a Customer type) looks something like:

```
{
  "name": "x.Customer",
  "friendlyName": "Customer",
  "pluralName": "Customers",
  "description": "Customers 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=\".../typeproperty\"",
      "method": "GET"
    },
    "recentOrders": {
      "rel": "collection",
      "href":
       "http://~/domainType/x.Customer/collections/recentOrders",
      "type": "application/json;profile=\".../typecollection\"",
      "method": "GET"
    },
    "blacklist": {
      "rel": "action",
      "href":
        "http://~/domainType/x.Customer/actions/blackList",
      "type": "application/json;profile=\".../typeaction\"",
      "method": "GET"
    },
    ...
  ],
```

```
  "typeActions" : {
    "isSubtypeOf": {
      "rel" : "typeaction",
      "href":
"http://~/domainTypes/x.Customer/typeactions/isSubtypeOf/invoke",
      "method" : "GET",
      "type":
        "application/json;profile=\".../typeactionresult\"",
      "arguments" : {
        "supertype" : {
          "href" : null
        }
    },
    "isSupertypeOf": {
      "rel" : "typeaction",
      "href":
"http://~/domainTypes/x.Customer/typeactions/isSupertypeOf/invoke",
      "method" : "GET",
      "type" :
        "application/json;profile=\".../typeactionresult\"",
      "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
links[rel=icon32]	(optional) link to an image to represent this type as an icon of 32x32 pixels
links[rel=icon16]	(optional) link to an image to represent this type as an icon of 16x16 pixels
links[rel=help]	(optional) link to a media resource providing help about the type

JSON-Property	Description
name	the fully qualified class name (or JSON type, if there is an equivalent)
friendlyName	the singular form of the type, as would be suitable for rendering in a UI. The value should be internationalized, §A2.16.
pluralName	the plural form of the type, as would be suitable for rendering in a UI. The value should be internationalized, §A2.16. The implementation should generate a value if need be.
description	a description of the type, eg to render as a tooltip. The value should be internationalized, §A2.16.
isService	indicates whether the type is a domain service or not
typeActions	map of type action invocation resources, §26.
members	map of links to resources representing a description of domain a object property §D22.1, a domain object collection §D22.2 or a domain object actions §D23.2.
extensions	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": "icon32",
    "href": "http://~/images/Customer-32x32.jpg",
    "type": "image/jpg",
    "method": "GET"
  },
  {
    "rel": "icon16",
    "href": "http://~/images/Customer-16x16.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.

22 DOMAIN 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:

```
/domainTypes/{domainType}/properties/{propertyId}
```

where:

- {domainType} is either
  - the fully qualified type name, or
  - is a built-in JSON type
- {propertyId} identifies the property.

22.1 HTTP GET

Obtain a representation that describes a domain property within the metamodel.

22.1.1 GET Request

22.1.1.1 Query String

- none

22.1.1.2 Headers

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

22.1.1.3 Body

- N/A

22.1.2 GET Successful Response

22.1.2.1 Status Code

- 200 "OK"

22.1.2.2 Headers

- **Content-Type**
  - application/json;profile="…/typeproperty"
- **Cache-Control**:
  - max-age: 86400 (1 day)

22.1.2.3 Body

As per §22.2.

22.1.3 GET Not found Response

22.1.3.1 Status Code

- 404 "Not found"

22.1.3.2 Headers

- **Warning**:
  - No such domain type {domainType}.
  - No such domain property {propertyId} in domain type {domainType}

22.1.3.3 Body

- empty

22.2 Representation

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

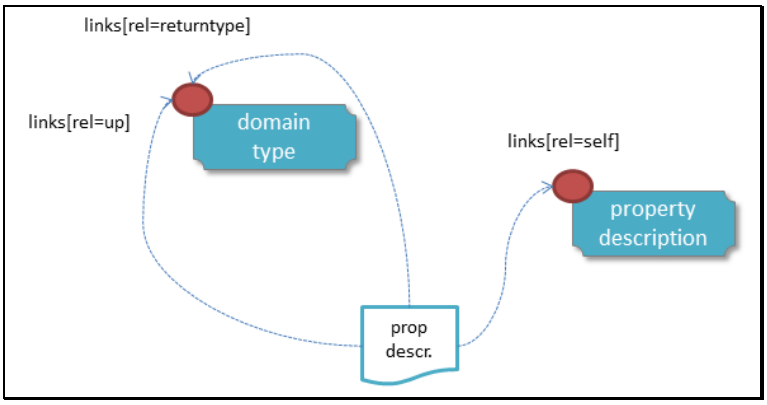

FIGURE 17: DOMAIN PROPERTY COLLECTION REPRESENTATION

The JSON representation (for the Order's deliveryOptions property) looks something like:

```
{
  "id": "deliveryOptions",
  "friendlyName": "Delivery Options",
  "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://~/domainTypes/x.Order",
      "type": "application/json;profile=\".../domaintype\"",
      "method": "GET"
    }, {
      "rel": "returntype",
      "href":
        "http://~/domainTypes/string",
      "type": "application/json;profile=\".../domaintype\"",
      "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 property
friendlyName	the property name, formatted for rendering in a UI. The value should be internationalized, §A2.16.
description	a description of the property, eg to render as a tooltip. The value should be internationalized, §A2.16.
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 as to the relative order to display each member
format	for properties returning a string value, indicates how to interpret that value §A2.5.
links[rel=up]	link to the domain type which owns this property

JSON-Property	Description
links[rel=returntype]	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.

23 DOMAIN 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:

```
/domainTypes/{domainType}/collections/{collectionId}
```

where:

- {domainType} is either
  - the fully qualified type name, or
  - is a built-in JSON type
- {collectionId} identifies the collection.

23.1 HTTP GET

Obtain a representation of a description of a domain collection, 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=… /typecollection

23.1.1.3 Body

- N/A

23.1.2 GET Response

23.1.2.1 Status Code

- 200 "OK"

23.1.2.2 Headers

- **Content-Type**
  - application/json;profile="…/typecollection"
- **Cache-Control**:
  - max-age: 86400 (1 day)

23.1.2.3 Body

As per §23.2.

23.1.3 GET Not found Response

23.1.3.1 Status Code

- 404 "Not found"

23.1.3.2 Headers

- **Warning**:
  - No such domain type {domainType}.
  - No such domain collection {id} in domain type {domainType}

23.1.3.3 Body

- empty

23.2 Representation

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

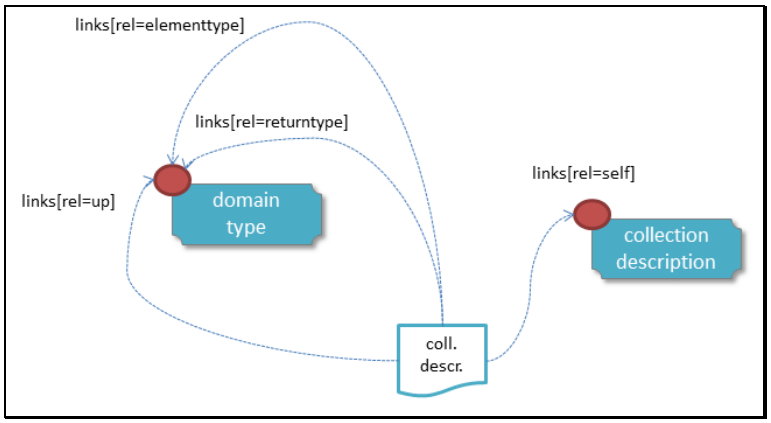

FIGURE 18: DOMAIN COLLECTION DESCRIPTION REPRESENTATION

The JSON returned representation (for the Order's items collection) would 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://~/domainTypes/x.Order",
      "type": "application/json;profile=\".../domaintype\"",
      "method": "GET"
    },
    {
      "rel": "returntype",
      "href": "http://~/domainTypes/list",
      "type": "application/json;profile=\".../domaintype\"",
      "method": "GET"
    },
    {
      "rel": "elementtype",
      "href": "http://~/domainTypes/x.OrderItem",
      "type": "application/json;profile=\".../domaintype\"",
      "method": "GET"
    },
    {
      "rel": "help",
      "href":
         "http://~/videos/training/Order-items-explained.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 collection
friendlyName	the collection name, formatted for rendering in a UI. The value should be internationalized, §A2.16.
pluralForm	the pluralized form of the element type within the collection/list. The value should be internationalized, §A2.16.
description	a description of the collection, eg to render as a tooltip. The value should be internationalized, §A2.16.

JSON-Property	Description
memberOrder	a presentation hint as to the relative order to display each member
links[rel=up]	link to the domain type which owns this property
links[rel=returntype]	link to the domain type for list or for set.
links[rel=elementtype]	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/properties to "**links**" and "**extensions**" as they require.

24 DOMAIN 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. A client can inspect all query-only domain service actions and make them available in the UI whenever an instance of their return type is required (as a property value, to add to collection or as an action argument)

The endpoint URL for this resource is:

```
/domainTypes/{domainType}/actions/{actionId}
```

where:

- {domainType} is either
  - the fully qualified type name, or
  - is a built-in JSON type
- {actionId} identifies the action.

24.1 HTTP GET

Obtain a representation of a description of a domain action, 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=".../typecollection"

24.1.1.3 Body

- N/A

24.1.2 GET Response

24.1.2.1 Status Code

- 200 "OK"

24.1.2.2 Headers

- **Content-Type**
  - application/json;profile="…/typecollection"
- **Cache-Control**:
  - max-age: 86400 (1 day)

24.1.2.3 Body

As per §24.2.

24.1.3 GET Not found Response

24.1.3.1 Status Code

- 404 "Not found"

24.1.3.2 Headers

- **Warning**:
  - No such domain type {domainType}.
  - No such domain action {actiond} in domain type {domainType}

24.1.3.3 Body

- empty

24.2 Representation

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

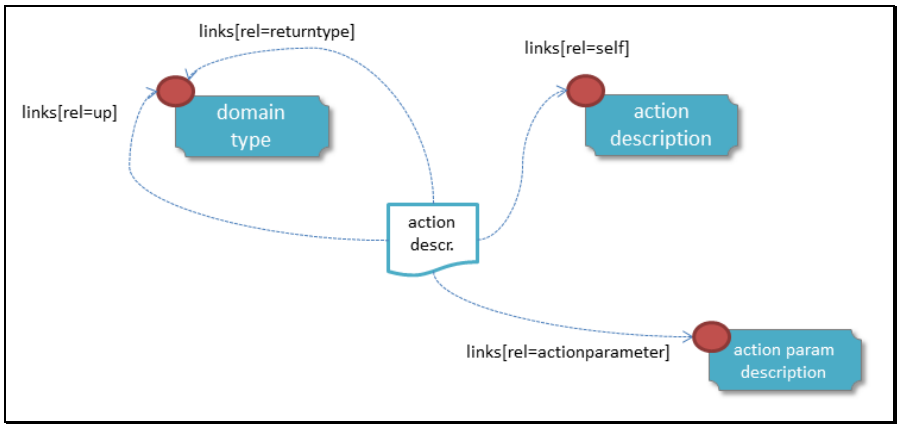

FIGURE 19: DOMAIN ACTION DESCRIPTION REPRESENTATION

The JSON representation of the domain action description (for the Order's submit action), it would look something like:

```
{
  "id": "submit",
  "friendlyName": "Submit",
  "description": ...,
  "hasParams": true,
  "memberOrder": 5,
  "links": [ {
      "rel": "self",
      ...
    }, {
      "rel": "up",
      "href": "http://~/domainTypes/x.Order",
      "type": "application/json;profile=\".../domaintype\"",
      "method": "GET"
    },
    {
      "rel": "returnType",
      "href": "http://~/domainTypes/x.OrderReceipt",
      "type": "application/json;profile=\".../domaintype\"",
      "method": "GET"
    },
    {
      "rel": "help",
      "href": "http://~/videos/training/Order-submit.mpg",
      "type": "audio/mpeg",
      "method": "GET"
    },
    ...
  ],
  "parameters": {
    "ship": {
      "rel": "actionparam ",
      "href": "http://~/domainTypes/x.Order/submit/params/ship",
      "type":
       "application/json;profile=\".../typeactionparameter\"",
      "method": "GET"
    },
    "rush": {
      "rel": "actionparam ",
      "href": "http://~/domainTypes/x.Order/submit/params/rush",
      "type":
       "application/json;profile=\".../typeactionparameter\"",
      "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

JSON-Property	Description
friendlyName	the action name, formatted for rendering in a UI. The value should be internationalized, §A2.16.
pluralForm	(optional); for actions returning collections, is the pluralized form of the element type within the collection/list. The value should be internationalized, §A2.16.
description	a description of the action, eg to render as a tooltip. The value should be internationalized, §A2.16.
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=returntype]	link to the action's return type
links[rel=elementtype]	(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 §25
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.

25 DOMAIN ACTION PARAMETER DESCRIPTION RESOURCE

The domain action parameter description resource represents a description 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:

```
/domainTypes/{domainType}/actions/{actionId}/params/{paramName}
```

where:

- {domainType} is either
  - the fully qualified type name, or
  - is a built-in JSON type
- {actionId} identifies the action
- {paramName} is the named action parameter

25.1 HTTP GET

Obtain a representation of a description of a domain action parameter, 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=".../typeactionparameter"

25.1.1.3 Body

- N/A

25.1.2 GET Response

25.1.2.1 Status Code

- 200 "OK"

25.1.2.2 Headers

- **Content-Type**
  - application/json;profile=".../typeactionparameter"
- **Cache-Control**:
  - max-age: 86400 (1 day)

25.1.2.3 Body

As per §25.2.

25.1.3 GET Not found Response

25.1.3.1 Status Code

- 404 "Not found"

25.1.3.2 Headers

- **Warning**:
  - No such domain type {domainType}.
  - No such domain action { actionId} in domain type {domainType}
  - No such parameter name

25.1.3.3 Body

empty

25.2 Representation

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

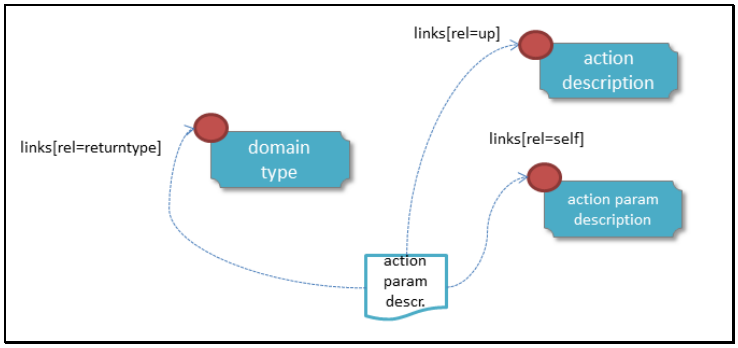

FIGURE 20: DOMAIN ACTION PARAMETER DESCRIPTION REPRESENTATION

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://~/domainTypes/x.Order/actions/submit",
      "type": "application/json;profile=\".../typeaction\"",
      "method": "GET"
    }, {
      "rel": "returntype",
      "href": "http://~/domainTypes/string",
      "type": "application/json;profile=\".../domaintype\"",
      "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 (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. The value should be internationalized, §A2.16.
description	a description of the action parameter, eg to render as a tooltip. The value should be internationalized, §A2.16.
optional	indicates whether the action parameter is optional

JSON-Property	Description
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.
format	for action parameters returning a string value, indicates how to interpret that string value §A2.5.
links[rel=up]	link to the action which owns this parameter
links[rel=returntype]	link to the action parameter's return type
links[rel=help]	(optional) link to a media resource providing help about the action parameter
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 or json-properties as they require.

26 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. However, (whereas the object action invoke resource can be used to invoke any domain object action), the type actions available to be invoked are explicitly specified.

Restful Objects defines the following type actions.

Type action	See	Description
isSubtypeOf()	§26.1	to determine if a domain type is a subtype of (or the same as) a supertype.
isSupertypeOf()	§26.2	to determine if a domain type is a supertype of (or the same as) a subtype.

The endpoint URL for this resource is:

```
/domainTypes/{domainType}/typeactions/{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.

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 (eg a list §B11, a scalar §B12, or a domain object, §C14).

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

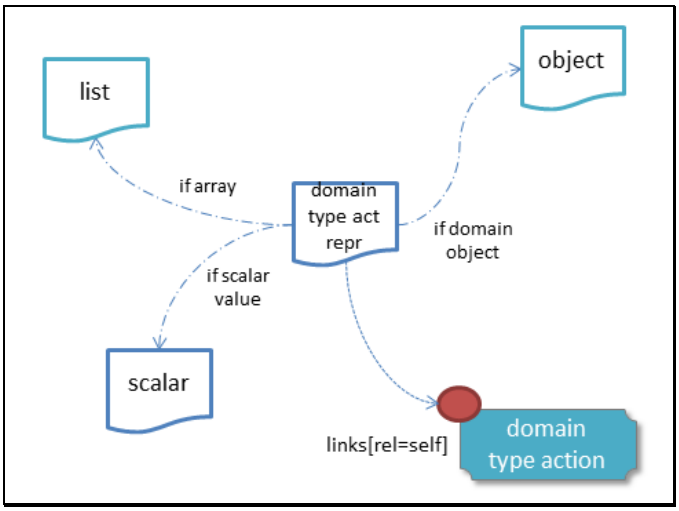

FIGURE 21: DOMAIN TYPE ACTION REPRESENTATION

The representations returned by type actions follow the format:

```
{
  "links" : [ {
    "rel" : "self",
    "href" : "http://~/domainTypes/x.Customer/typeactions/...
/invoke",
    "method" : "GET",
    "type" :
"application/json;profile=\"urn:org.restfulobjects/typeactionsresul
t\"",
    "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.

26.1 HTTP GET isSubtypeOf()

Obtain a representation resulting from the invocation of the isSubtypeOf() domain type action. This is the reciprocal of the isSupertypeOf() type action §26.2.

26.1.1 GET Request

26.1.1.1 URL and Query String

The endpoint URL for this resource is:

```
/domainTypes/{domainType}/typeactions/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 the simple style of arguments §A2.8.1 are supported by the implementation, then the query string is simply:

```
?supertype=xxx
```

where xxx is the fully qualified type name of the supertype.

Formal Style

If query arguments are specified using the formal style (§A2.8.2) then the supertype should be specified as a map and then the map encoded as a URL (§A2.8.2.5).

For example:

```
{
  "supertype": {
    "value": {
      "href": "http://~/domaintypes/x.BasketOwner",
    }
}
```

Note that the value should be a link to the supertype, rather than simply the name of the supertype.

26.1.1.2 Headers

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

26.1.1.3 Body

- N/A

26.1.2 GET Response

26.1.2.1 Status Code

- 200 "OK"

26.1.2.2 Headers

- **Content-Type**
  - application/json;profile=".../typeactionresult"
- **Cache-Control**:
  - max-age: 86400 (1 day)

26.1.2.3 Body (representation)

The JSON representation returned The JSON representation returned is a typeactionresult representation (as described earlier) with a scalar representation §B12 inlined.

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://~/domainTypes/x.Customer/typeactions/isSubtypeOf/invoke",
    "method" : "GET",
    "type" :
"application/json;profile=\"urn:org.restfulobjects/typeactionresult
\"",
    "arguments" : {
      "supertype" : {
        "href" :
          "http://~/domainTypes/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.

26.1.3 GET Not found Response

26.1.3.1 Status Code

- 404 "Not found"

26.1.3.2 Headers

- **Warning**:
  - No such domain type {domainType}.
  - No such domain type action { actionId} in domain type {domainType}
  - No such super/subtype

26.1.3.3 Body

empty

26.2 HTTP GET isSupertypeOf()

Obtain a representation resulting from the invocation of the isSupertypeOf() domain type action. This is the reciprocal of the isSubtypeOf() type action §26.1.

26.2.1 GET Request

26.2.1.1 URL and Query String

The endpoint URL for this resource is:

```
/domainTypes/{domainType}/typeactions/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 the simple style of arguments §A2.8.1 are supported by the implementation, then the query string is simply:

```
?subtype=xxx
```

where xxx is the fully qualified type name of the subtype.

Formal Style

If query arguments are specified using the formal style (§A2.8.2) then the subtype should be specified as a map and then the map encoded as a URL (§A2.8.2.5).

For example:

```
{
  "subtype": {
    "value": {
      "href": "http://~/domaintypes/x.Customer",
    }
  }
}
```

Note that the value should be a link to the subtype, rather than simply the name of the subtype.

26.2.1.2 Headers

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

26.2.1.3 Body

- N/A

26.2.2 GET Response

26.2.2.1 Status Code

- 200 "OK"

26.2.2.2 Headers

- **Content-Type**
  - application/json;profile=".../typeactionresult"
- **Cache-Control**:
  - max-age: 86400 (1 day)

26.2.2.3 Body (representation)

The JSON representation returned The JSON representation returned is a typeactionresult representation (as described earlier) with a scalar representation §B12 inlined.

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://~/domainTypes/x.BasketOwner/typeactions/isSupertypeOf/invok
e",
    "method" : "GET",
    "type" :
"application/json;profile=\"urn:org.restfulobjects/typeactionresult
\"",
    "arguments" : {
      "supertype" : {
```

```
        "href" :
          "http://~/domainTypes/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.

26.2.3 GET Not found Response

26.2.3.1 Status Code

- 404 "Not found"

26.2.3.2 Headers

- **Warning**:
  - No such domain type {domainType}.
  - No such domain type action { actionId} in domain type {domainType}
  - No such super/subtype

26.2.3.3 Body

empty

E

PATTERNS AND PRACTICES

27 HATEOAS vs Web APIs

Comparing two styles of distributed system.

27.1 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*. Just as when using a web browser you can click through links to get to the next page, so should a REST client be able 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; 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.

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 actions can return references to other objects. Links are only included in representations if it makes sense for the client to follow them.

27.2 Web APIs (Templated URIs)

Some REST 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 it is performant (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 should be prepared to recover if the link is no longer valid).

Even so, many developers prefer a system designed following a more API-based approach works, defining a set of well-known *templated URIs*, documented in the form:

```
path/{pathParam1}/pathPart/{pathParam2}/…
```

So long as the client has arguments for each of the {pathParam}'s then it can directly contruct the URL and invoke it.

It is debatable whether this style of system should be called REST; a better name is probably a(n HTTP-based) web API. Nevertheless it is popular, and 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.

28 PERSONAL VS SHARED STATE

Different types of resource state, what is not resource state, and discussion of the "domain-model rest anti-pattern" paper.

28.1 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 §29).

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 to client to navigate between entities, either as a result of following a property or collection, 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 (lnk 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 (§27.1) of RESTful systems.

28.2 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 (credit card, delivery options and so on).

Restful Objects does not specify how implementations should distinguish between a resource that holds personal state and one that holds shared state. One possible approach though would be for implementations to look for a reserved/annotated "UserId" property 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 returned as a resource if requested by any other user.

28.3 Application State

Personal state, discussed above, is not the same as application state, though the distinction is subtle. In a blog post from 2008[31], 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.*

[31] http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven#comment-744 .

At first glance one might consider that this definition does not afford for 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 that consuming its representation generates on the client. 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.

28.4 Domain Model Resources an Anti-Pattern?

Some REST practitioners argue[32] 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.

28.4.1 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 domain is complex and the users needs explicit assistance in order to navigate it. One of the earliest examples was Microsoft Money 2000[33], which took users through various common-place financial book-keeping tasks.

[32] And they argue quite strongly; see for example http://java.dzone.com/articles/domain-model-rest-anti-pattern.

[33] http://msdn.microsoft.com/en-us/library/ms997506.aspx

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[34].

28.4.2 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.

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 structures 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 of those entities made directly available for the user to invoke as they see fit. This is not a CRUD system; the behaviour on the entities is every bit as rich as the behaviour exposed by a use case resource.

In practice, most systems 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 styl.

A related approach is to start with a deductive system, and then to add 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.

[34] http://en.wikipedia.org/wiki/Application_posture

29 CLIENT VS SERVER EVOLUTION

Why content negotiation may be needed, and when it is not needed.

29.1 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.

29.2 Content Negotiation

The standard solution to this is to version representations, with media types as the identifier for a particular version of a representation. 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 supports conneg through the **x-ro-domaintype** media type parameter §A2.4.2. The server indicates the domain type of each object representation it serves, and the client can specify the domain types that it understands.

In some circumstances it may be possible to version domain entities, eg providing a v2.Customer and a v3.Customer. Doing this effectively generally requires some sort of abstraction layer closer to the database, for example persisting to an RDBMS that supports views and "instead-of" triggers.

for a more common approach to versioning is to use view models. Thus, one client may consumev2.CustomerViewModel, while another may consume v3.CustomerViewModel. Each of these view models delegates to an underlying Customer entity, but doesn't expose it directly. This allows the Customer domain entity to evolve as needed without breaking existing clients.

Restful Objects does not mandate how implementations support conneg. However a code sketch of how an implementation might accomplish this *is* described in § 31.2.

29.3 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 versons 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.

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 which 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 *application/json;profile="urn:org.restfulobjects/xxx"* representations, and as such would be able to render domain objects, lists, action prompts and so on.

To summarize:

- If the client and server are developed together, then the default media types may be used.
- If the server and client are developed independently, but the client is generic, then again the default media types may be used.
- If the server and client are independent and the client makes hard-coded assumptions about the representations that it consumes, then versioned media types (typically through view models) and content negotiation should be used.

30 DEALING WITH UNTRUSTED CLIENTS

How implementations can avoid important information leaking out to untrusted clients

30.1 Securing HREFs

One possible deployments for Restful Objects systems is a Restful Objects server on the internet, serving up JSON to a Javascript "single-paged 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, eg 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 object identifier OID part ("CUS-12345") is opaque and therefore implementation-specific. Therefore, an implementation can protect itself from hacking by ensuring that the OID 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 effect this.

31 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

31.1 Making View Models Addressable

As discussed in §A2.2, an addressable view model is a view models that, like a persistent domain entity, has an addressable resource, enabling it to expose actions.

This implies that it has an oid and some sort of server-side state. However, unlike a persistent domain entity, there is no database row, and so the state must be recreated on the fly.

There are several ways that a framework implementation of Restful Objects could do this, but a simple solution is to derive all state from the oid value obtained from the resource.

For example, the representation of a CustomerViewModel could contain the following link to an action:

```
http://~/objects/CVM-123/actions/findRecentOrders
```

The implementation could parse the oid "CVM-123" and determine from its metamodel that "CVM" corresponded to CustomerViewModel. This information could, for example, have been gleaned by the view model implementing an interface:

```
public interface AddressableViewModel {
  String getOidPrefix();
  ...
}
```

and:

```
public class CustomerViewModel implements AddressableViewModel {
    public String getOidPrefix() { return "CVM"; }
}
```

The implementation could next instantiate the view model, and then pass in the oid string. The new view model in turn would use parse the oid in order to recreate the rest of its state. For example, the "123" part of the oid "CVM-123" might correspond to a Customer object which it would look up using an injected Customers repository domain service.

What about the creation of the oid in the first place? Well, this could be the same process, but in reverse, with the view model providing its oid.

Both the setting and getting of the oid could be defined by the interface.

Putting the above together:

```
public interface AddressableViewModel {
  String getOidPrefix();
  String getOidSuffix();
  void setOidSuffix(String oidSuffix);
}
```

where:

- getOidPrefix() is used by the implementation to instantiate the correct view model implementation
- getOidSuffix() is used by the implementation to manufacture URLs in the representations
- setOidSuffix() is used by the view model to recreate its state

An alternative but equivalent approach would be to use annotations/attributes, eg:

```
@AddressableViewModel("CVM")
public class CustoemrViewModel {

  String getOidSuffix() {
    return "" + getCustomer().getId();
  }
  public void setOidSuffix(String oidSuffix) {
    int id = Integer.parseInt(oidSuffix);
    setCustomer(customerRepository.findById(id));
  }

  ...

  @Hidden
  public Customer getCustomer() { ... }
  public void setCustomer(Customer customer) { ... }

  // customerRepository injected
  ...
}
```

From the developers' perspective, since the point of a view model is to provide a stable API for clients (as exposed by the **x-ro-domainType** 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.

31.2 Supporting Conneg

As discussed in §29, one reason for using view models is to enable the decoupled evolution of client and server. Each view model is versioned; the client uses content negotiation to request the version of the view model that it understands by specifying the **x-ro-domain-type** parameter in the request **Accept** header:

```
Accept: application/json;
  x-ro-domain-type=
    http://~/domainTypes/x.viewmodels.v2.CustomerViewModel
```

However, since only the Restful Objects framework implementation has access to the **Accept** header, a mechanism is required by which the framework can indicate which view model to return.

It is easy enough for the framework implementation to parse the **x-ro-domain-type** and determine the corresponding domain type (eg 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?

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 = converterRegistory.find(
  objectToReturn.getClass(), requiredReturnType);

if (converter == null) { ... throw a 406 ... }
return converter.convert(objectToReturn, requiredReturnType);
```

32 FAQs

32.1 "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?"

The media types for the representations defined by Restful Objects indicate that a representation is of an object, a property, a collection or an action, but they do *not* indicate a particular type of object (like Customer). It is therefore possible to write a fully generic, purely RESTful client, driven purely from hypertext, if you so wish.

Alternatively, you can use application-defined media types so that clients can use conneg to request specific representations of domain objects. That these domain objects may be entities is an implementation concern; §28 provides some background on how frameworks may choose to address this point.

It is also worth noting that close coupling between client and server may not be an issue; an application team may control the deployment of both and has chosen to use REST to connect them together for other reasons. In this case domain entities can be exposed without any need to create a projection or application-defined media types. An obvious example is a client written using code-on-demand (ie Javascript), where the client pulls representations from multiple resources to support specific use cases[35].

32.2 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).

[35] Code-on-demand can also be used to develop quite sophisticated generic clients, using Javascript templating engines.

As noted in §28, 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.

32.3 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.

