RESTful Web Services and HTTP: Standards and Sources

Find this document at: https://restful.ws

A thoroughly linked reference for producers of articulate, RESTful web services and API portfolios

version 2021.03.23-00

Comments and suggestions: Colin Jaccino <*colin@restful.ws*>

Timeline

1989 Conception of the World Wide Web

Tim Berners-Lee invents the World Wide Web, providing uniform resource addressing (URIs), a hypertext document format (HTML), and a protocol for distributing hypertext documents (HTTP). Early implementations were available in 1991, and within a few years, the Web becomes the Internet's primary application.

1994 World Wide Web Consortium (W3C) Founded

1996 Initial Specification. rfc1945: HTTP/1.0

1997 rfc2068: HTTP/1.1

Per the 2017 Reflections paper, REST was "born as a byproduct of the collaboration between Fielding and Nielsen while working on the HTTP specifications, pruning HTTP/1.0 to the essential bits and evaluating various ideas... for a future HTTP/1.1".

1999 rfc2616: HTTP/1.1 Update

A relatively minor revision, rfc2616 provided the HTTP standard for fifteen years. Though the document did not mention REST, the terminology for resource, representation, naming (URIs) articulate a resource-oriented architecture.

2000 Coining REST: Fielding's Dissertation

Architectural Styles and the Design of Network-based Software Architectures, defined REST as an architectural style composed composed of constraints that would enable a scalable, distributed hypermedia network.

Principled Design of the Modern Web Architecture

A reiteration of much of Fielding's work presented via the International Conference on Software Engineering (ISCE). Provided additional articulation of REST and drove additional attention to the architectural style.

2004 W3C: Architecture of the World Wide Web, Volume One

The W3C's recommendation for Web Architecture provided substantial depth on the treatment Internet resources, from their identification (URIs), to interaction, and representation. A significant step forward in guidance for practical web service design.

2008 Untangled: REST APIs must be Hypertext-driven

In a high profile critique of a non-RESTful web service, Roy Fielding clarifies numerous topics on REST, both in his blog article and on the subsequent Q&A.

Richardson Maturity Model

Following the release of his book, "RESTful Web Services", Leonard Richardson presents the "Maturity Heuristic" at QCon 2008. His ideas, popularly known as Richard Maturity Model, spread quickly as an easy way to measure a Web Service's maturity relative to key RESTful constraints.

2014 HTTP/1.1 Update (rfc7230-7235)

Under a new working group, httpbis, again led by Fielding, the IETF publishes a suite of standards covering numerous aspects of HTTP at much greater depth than the standard it replaced, rfc2616. The HTTP Semantics specification, rfc7231, establishes REST and resource orientation as HTTP's primary paradigm.

2017 Reflections on the REST Architectural Style

With two decades of experience with REST and the Web's evolution, Fielding, Taylor, et al revisit early RESTful concepts, unanticipated challenges, and key lessons in the paper "Reflections on the REST Architectural Style and "Principled Design of the Modern Web Architecture".

OpenAPI 3 Released

The OpenAPI Initiative made the first release of OpenAPI 3, an industry standard for describing HTTP-based web services and APIs. Based on an earlier format,- from SmartBear/Swagger, OpenAPI would become the most popular format for design-first API development.

Future HTTPbis and Ongoing Standards Development

The IETF HTTPbis working group continues to update HTTP standards. Current work is expected to more clearly separate semantics and syntax. Semantics will be common across different versions of the transports, HTTP/1.1, HTTP/2, and the anticipated HTTP/3 based on QUIC.

Securing and Managing Access

Security

More facets of security for Web Services than can be accounted for. This section provides issues, terms.

- Protecting confidentiality,integrity of message and channel: TLS, MTLS, JWT and message signing
- Defending against abuse patterns: CORS, Reverse proxies and API Gateways, Input validation, object schema validation, quotas and throttling, OWASP Top Ten
- Authorization and Policy: OAuth scopes, OpenID Connect identity tokens, JWT, XACML

Identity

OAuth2 is being simplified. See drafts for OAuth 2.1, OAuth 2 Best Practices and WG related.
When designing Authorization, be sure to consider consider the following:

- **The Authorized Party**: Who or what will be authorized to use it? What is it that has the credentials? A person? A client app? A device?
- **Roles and Entitlements**: What differences will there be in what a client can do?
- **Implementation**: Where is authorization functionality enforced?

OAuth2 & OpenID Connect

IANA OAuth Params

rfc6749	The OAuth 2.0 Authorization Framework
rfc6750	OAuth 2.0 Bearer Token Usage
rfc6819	OAuth 2.0 Threat Model, Security Considerations
rfc7009	OAuth 2.0 Token Revocation
CORE	OpenID Connect Core 1.0
DISC	OpenID Connect Discovery 1.0
SESSION	OpenID Connect Session Management 1.0
FORM	OAuth 2.0 Form Post Response Mode

JWT / JSON Object Signing & Encryption

rfc7165	Use Cases, Reqs for JSON Object Signing and Encryption (JOSE)	
rfc7515	JSON Web Signature (JWS)	
rfc7516	JSON Web Encryption (JWE)	
rfc7519	JSON Web Token (JWT)	IANA JWT
rfc7520	Examples of Protecting Content Using JSON Object Signing and Encryption (JOSE)	

Legend

rfc5789	Source Callout; Usually for non-core sources
Safe	Safe HTTP Method; will make no change
Idempotent	Idempotent HTTP Method; change only for first request
Cacheable	Cacheable HTTP Method
IANA Registry	Indicates a relevant IANA registry
IETF Supplement	A relevent supplemental IETF Source

REST's Foundations

The REST concept ***preceded*** standardization of the World Wide Web. The style was designed to enable a scalable, distributed, network of hypermedia resources.

Fielding and team developed and used the REST concept as a way to test choices being made as HTTP was formalized. Thus, HTTP was *made for REST*. REST does not require HTTP, but HTTP is the most well-known use of the architecture style.

HTTP is flexible, making it possible to violate REST's constraints. When Web Services do so, they become non-RESTful and benefit from fewer of the advantages REST provides.

Designers of RESTful Web Services and APIs should be informed of REST's constraints, trade-offs, and costs of deviating.

Fielding's Dissertation (2000)
Architectural Styles and the Design of Network-based Software Architectures
REST's formal introduction. The paper not only defines REST, but shows how it is derived from principles as an architectural style.

Fielding's ISCE Paper (2000)
Principled Design of the Modern Web Architecture
A restatement of Fielding's dissertation; packaged more accessibly

W3C: Architecture of the World Wide Web, Volume One (2004)
Significant contribution to the nature of resources, URIs, data formats.

Fielding et al: Retrospective on REST (2017)
Reflections on the REST Architectural Style and "Principled Design of the Modern Web Architecture"
Review of the 2000 ISCE paper, as well as successes and challenges for REST as an architecture style.

6 REST Constraints

- **Client-Server**
  Separation of concerns; client/server independence
- **Stateless**
  Client state cannot be stored on the server
  Resource and client application state are independent
- **Cache**
  Improves network efficiency
- **Uniform Interface**
  Four constraints define RESTful uniform interface
  Enables component decoupling
- **Layered System**
  Narrowly-defined, uncoupled layers enable Internet scale
- **Code-on-Demand**
  Flexibility. Reduces client pre-implementation burden

Four Uniform Interface Constraints

- **Identification of Resources**
  Resource Identification: The key abstraction of information in REST
- **Manipulation of Resources through Representations**
  Representations capture current or intended state.
  Clients indicate intent toward a resource
- **Self-descriptive Messages**
  Interactions stateless between messages
  Standard methods and media types provide semantics
  Responses explicitly indicate cacheability
- **Hypermedia as the Engine of Application State (HATEOAS)**
  Application state belongs to client. Resource state belongs to server.
  Links in representations and metadata offer possible interactions
  Client application state influenced by interactions with resources
  Client state never stored on server.

REST Data Elements

resource	Conceptual target of a URI or reference
resource identifier	Provides a resource name or distinguishing ID. URI, URN, URL Having an identifier makes a resource addressible as the subject of a RESTful interaction
resource metadata	Information about a resource. Alternative URIs, the authoritative or canonical URI
representation	data composed as defined by a media type that conveys a resource in a given state, often the current or desired state
representation metadata	information about a representation. media type, time generated, entity tag, etc
control data	Information governing interpretation or processing of a request or response

Identifying Resources

IANA URI Schemes

Identify important things with *Uniform Resource Identifiers*, or **URIs**. URIs make them addressable. **URI**s come in many forms. Two special kinds of URIs are *Uniform Resource Names*, **URN**s, which name things, and *Uniform Resource Locators,* **URL**, which lead us to a place to interact with the resource.

REST prescribes that clients interact with resources by passing *representations* between Client and Server. To live clients receiving URIs should treat them opaque, not inferring meaning from path elements; true RESTful clients rely on hypermedia, not URI structure. Nevertheless, great APIs often have a consistent, well-concieved URI plan. If adoption is a goal, a more consumable URI plan can improve developer experience and first impressions during API selection.

Key URI Advice from W3C Web Architecture

- Global naming leads to global network effects.
- Assign distinct URIs to distinct resources.
- A URI owner SHOULD NOT associate arbitrarily different URIs with the same resource. URI aliasing is useful, but has downsides; use it sparingly.
- Agents making use of URIs SHOULD NOT attempt to infer properties of the referenced resource.

For API Producers

- Every URI path element is meaningful, identifies something
- URI scheme should reflect an underlying resource model
- URIs benefit from hierarchical organization
- Consider URI planning at the portfolio level

rfc3986	Uniform Resource Identifier (URI): Generic Syntax
rfc3987	Internationalized Resource Identifiers (IRIs)
rfc8615	Uniform Resource Names (URN)
rfc8820	URI Design and Ownership

HTTP Semantics

draft-ietf-httpbis-semantics

Still in development, the IETF's HTTP Semantics draft is the *single most informative document* on the use of HTTP. HTTP Semantics applies to HTTP/1.1 and /2, web sites and APIs.

HTTP Semantics covers:

- resources and resource identification
- representations and content
- expression of intent via HTTP Methods
- the meanings of the standard HTTP Status Codes
- headers and trailers
- authorization and security

Define your organization's web API and portfolio standards using this semantics draft or its authoritative predecessors.

The Semantics Draft will replace:

rfc2818	HTTP Over TLS
rfc7230	HTTP/1.1 Message Syntax and Routing (*)
rfc7231	HTTP/1.1 Semantics and Content
rfc7232	HTTP/1.1 Conditional Requests
rfc7233	HTTP/1.1 Range Requests
rfc7235	HTTP/1.1 Authentication
rfc7238	HTTP Status Code 308 (Permanent Redirect)
rfc7615	HTTP Authentication-Info and Proxy-Authentication-Info Response Headers
rfc7694	HTTP Client-Initiated Content-Encoding

HTTP Methods: Expressing Client Intent Toward Resources

Safe Idempotent Cacheable	GET	Request a Representation
Safe Idempotent Cacheable	HEAD	Request GET Metadata without Representation
Cacheable	POST	Request Processing of Representation Using Resource's Semantics
Idempotent	PUT	Create/Replace Target Resource State using Provided Representation
Idempotent	DELETE	Remove Association between URI and Resource
Safe Idempotent	OPTIONS	Request Communication Options for Resource
	CONNECT	Create Tunnel between Client and Target
Safe Idempotent	TRACE	Request Loop-back of Request Message
rfc5789	PATCH	Partial Update a Resource Representation

HTTP Response Status Codes

IANA HTTP Status Codes

1xx Informational

100	Continue	
101	Switching Protocols	

2xx Successful

200	OK	*Success, with message body*
201	Created	*New resource created, see Location header.*
202	Accepted	*Processing continuing. Noncommital. Asynchronous response may follow*
203	Non-Authoritative Information	
204	No Content	*Success, without message body*
205	Reset Content	
206	Partial Response	

3xx Redirection

300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect

5xx Server Error

500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

4xx Client Error

400	Bad Request	*General client request error*
401	Unauthorized	*Lacking client credentials*
402	Payment Required	
403	Forbidden	*Refused to fulfill request*
404	Not Found	*No representation found*
405	Method Not Allowed	
406	Not Acceptable	*No representation for Accept header*
407	Proxy Authentication Required	
408	Request Timeout	
409	Conflict	*Conflict with resource current state*
410	Gone	*Resource no longer available*
411	Length Required	*Missing Content-Length header*
412	Precondition Failed	
413	Payload Too Large	
414	URI Too Long	
415	Unsupported Media Type	
416	Range Not Satisfiable	
417	Expectation Failed	
422	Unprocessable Payload *rfc4918*	
426	Upgrade Required	
428	Precondition Required *rfc6585*	
429	Too Many Requests *rfc6585*	*Throttling and Too Busy*
431	Request Header Fields Too Large *rfc6585*	

Uniform Resource Identifier (URI)

scheme	authority	path	query	fragment
`https`	`restful.ws`	`/posters/reference`	`?current=true`	`#timeline`

`https://restful.ws/posters/reference?current=true#timeline`

HTTP Request

IANA HTTP Methods	rfc7231 The client's intent (**GET**) and URI	`GET /posters/reference?current=true`
	rfc7230 The authority or origin **host** for the resource	`Host: restful.ws`
IANA HTTP Authentication Schemes	rfc6750, rfc7235 Client **Authorization**	`Authorization: BEARER YN6Y8U9D89zEQTJTb2fJmHST`
	rfc7231 Client's **accept**able media types in the response	`Accept: application/pdf;q=0.8, text/html`

HTTP Response

IANA HTTP Status Codes	rfc7231 The **status** of the response.	`200 OK`
IANA HTTP Media Types	rfc7231 The **media type** of the payload	`Content-Type: application/pdf`
	rfc7230 The size of the payload in bytes	`Content-Length: 823402`
	rfc7231 The **date and time** of message origination	`Date: Mon, 15 Mar 2021 22:31:44 GMT`
	rfc7232 **Entity tag** (etag) opaque validator	`etag: "9cc9fac6dc7336979fb96b64f76c0d06"`
	rfc7230 **Via** indicating intermediate recipients	`via: 1.1 849381c36-cdn.example.com`
	rfc7230 The **payload** or **message body**	`(abbreviated) FbzWUdxuFdtFukf5KCNsWJwYRtBEDp1im-SP+fJhcy5BXOVcN3Ey0GMbUmJuuTXPCOT1EPw1V5vURqObY...`

Headers

Client Request Headers

Clients provide metadata to indicate message handling preferences, identify the client or user, indicate response representation acceptability, describe the request's payload, and enable validation of representations for caching and conditional scenarios

Server Response Headers

Servers control the resources. Response headers describe the representations or underlying resource, expected response handling, direct client handling or processing, and elaborate on interactions available for the resource.

Content Negotiation and Representation

Client Request Headers			Server Response Headers		
Accept	Acceptable media type for representation				
Content-Type	Representation media type	IANA Media Types	Content-Type	Representation media type	IANA Media Types
Content-Length	Payload size in bytes		Content-Length	Payload size in bytes	
Accept-Encoding	Acceptable encodings for representation				
Content-Encoding	Representation encoding	IANA Content Coding	Content-Encoding	Representation encoding	IANA Content Coding
Accept-Language	Acceptable language for representation's audience				
Content-Language	Representation's language	rfc5646 Language Tags	Content-Language	Representation's language	rfc5646 Language Tags
Content-Location	Alternate URI for the same representation		Content-Location	vvate URI for the same representation	
Transfer-Encoding	Encoding applied to representation for transfer		Transfer-Encoding	Encoding applied to representation for transfer	

Resource Metadata

Client Request Headers			Server Response Headers		
			Location	The location relevant to the resource	
			rfc8288 Link	Links associated with the resource	IANA Link Relations
			rfc8549 Sunset	Expected date of resource becoming unresponsive	
			draft Deprecation	Signals deprecation of a URI, supporting information	

Message Routing

Client Request Headers			Server Response Headers		
Host	The host authority for the target URI		rfc7239 Forwarded	Provides information lost in proxying rfc7239	IANA Forwarded
Connection	Desired control options for current connection		Connection	Desired control options for current connection	
Upgrade	Invitation to upgrade to another protocol		Upgrade	Indication of switched protocol	IANA HTTP Upgrade Tokens
Via	Indicates intermediaries and protocols		Via	Indicates intermediaries and protocols	

Caching, Preconditions, and Validation

Client Request Headers			Server Response Headers		
Cache-Control	Directives for caching	IANA Cache Directives	Cache-Control	Directives for caching	IANA Cache Directives
If-Modified-Since	Honor request if representation modified since		Age	Seconds since response generation or validation	
If-Unmodified-Since	Honor request if representation not modified since		Expires	Time after which response is considered stale	
If-Match	Honor request if representation exists or matches ETag		Etag	Entity tag; opaque validator unique to representation	
If-None-Match	Honor request if no representation or matching ETag		Last-Modified	Time the representation was most recently changed	
If-Range	Honor request for range if representation is unchanged		Vary	Elements of request that influenced representation	

Control and Control Data

Client Request Headers			Server Response Headers		
Expect	Indication of behaviors required to handle message		Date	Date of response message	
Max-Forwards	Maximum number of forwards by intermediaries		Retry-After	Advice to retry after given number of seconds	
TE	Acceptable transfer encodings				
Trailer	Indiciates presence of trailers		Trailer	Indicates presence of trailers	

Context

Client Request Headers			Server Response Headers		
From	Email Address of person behind user agent		Allow	Methods allowed for target resource	
Referrer	Resource URI providing target URI		Server	Software information for origin server	
User-Agent	User agent originating request				

Ranges

Client Request Headers			Server Response Headers		
Range	The portion of the representation requested		Accept-Ranges	Indication resource supports range requests	
			Content-Range	Indication of the range of a partial representation	

HTTP State (Cookies)

Client Request Headers			Server Response Headers		
rfc6265 Cookie	Client-stored state information provided by server		rfc6265 Set-Cookie	Set client-stored state information	

Controversial *Cookies provide a non-REST state management mechanism. RESTfully speaking, this architectural anti-pattern mixes application state (client-side) with resource state (server).*
Recommendation *Consider the Foundations sources addressing REST state management as you define your application's architecture. See 5.3.3 in Fielding's dissertation.*

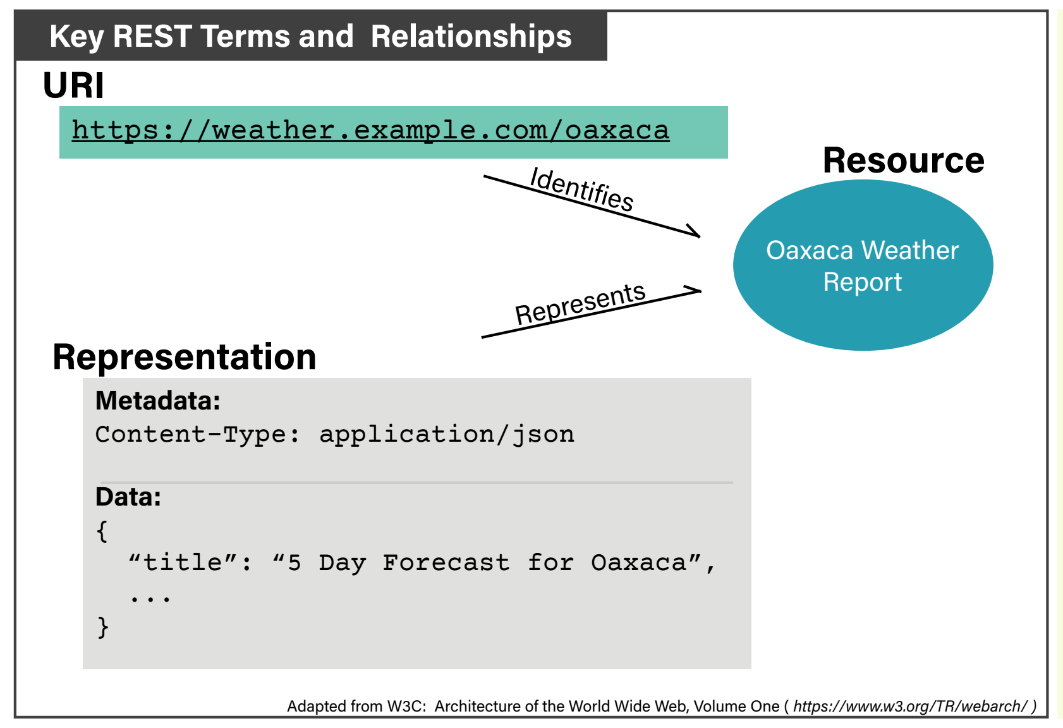
Authentication and Authorization

Client Request Headers			Server Response Headers		
Authorization	The user agent's authentication information for the origin		WWW-Authenticate	Authentication challenge(s)	IANA HTTP Authentication Schemes
			rfc6265 Authentication-Info	Information about the acceptance of credentials	
Proxy-Authorization	The user agent's authentication information for the proxy		Proxy-Authenticate	Proxy authentication challenge(s)	IANA HTTP Authentication Schemes
			rfc6265 Proxy-Authentication-Info	Proxy authentication challenge(s)	

Security and Privacy

Client Request Headers			Server Response Headers		
			rfc6797 Strict-Transport-Security	User agent security directives	
			fetch X-Content-Type-Options	"nosniff"; prevents media type sniffing	
			CSP2 Content-Security-Policy	User agent requirements for secure content processing	

Key REST Terms and Relationships



Metadata:

```
Content-Type: application/json
```

Data:

```
{
  "title": "5 Day Forecast for Oaxaca",
  ...
}
```

Adapted from W3C: Architecture of the World Wide Web, Volume One (*https://www.w3.org/TR/webarch/*)

Resource Oriented Architecture (ROA)

REST** is about **resources.

Many designers inadequately recognize that any RESTful service is organized around resources. While a URI can address anything, what is addressed must still be *a thing*. URI elements whose meaning is to *do something*, rather than *be something*, often indicate non-RESTfulness.

As Leonard Richardson presented and Martin Fowler elaborated on in the *Richardson Maturity Model*, REST web services have a set of traits.

1. Resource Orientation in their URIs
2. Faithful use of HTTP Methods
3. Hypermedia Controls

Fielding has stated that without hypertext or hypermedia, the service is not RESTful. Other styles include remote procedure call (RPC), plain-old-XML (POX) or (POJ), SOAP and GraphQL.

Defining and Representing Resources

Model the Resource

Before you can represent it, you must first *understand the resource*. What is it? What are its attributes? What can it do, and what can be done to it? When you ***model** your resource*, you create an abstract concept that governs its interactions, related resources, and representations. In some cases, UML Class Diagrams may be an appropriate way to define a resource's attributes, possible interactions, and relationships to other resources.

Choose Representation Format(s)

Representations offer the state of a resource using one or more media type. How might the resource be represented? Structured data formats, such as JSON or XML, are common for APIs. Image formats can depict physical or rendered states. Audio formats might represent a song, speech, or story. HTML for web pages. The IANA Media Types registry provides standardized media types. These media types will define how the resource is represented as client and server trade its current and desired states. A rich web service may support many media types for a resource.

Common Media Types for Web Pages IANA Media Types
text/html, application/javascript, image/*, text/css

Common Media Types in APIs IANA Media Types
application/xml, application/json
Custom formats with +xml and +json suffixes

Define Schema

Many APIs use a standard structured format, such as JSON (rfc8259), for representations. These media types provide the client what it needs to validate the syntax of the representation. They do not provide ontology or semantics. The designer must define a **schema**, a concrete description of how to represent the resource using a chosen format. Draw from your modeling work and consider how that schema will represent your resource through the interactions your service will enable.

Use JSON Schema to specify JSON representations. You might use OpenAPI 3.0 to define your resource representations and HTTP methods together. With OpenAPI 3.1 adopting JSON Schema, JSON Schema has become even more useful..

Consider defining your own media types to provide both semantics and syntax together.

Just Syntax
application/json

Semantics and Syntax
application/problem+json

rfc6838	Media Type Specification And Registration
rfc6839	Additional Media Type Structured Syntax Suffixes
rfc7303	XML Media Types

Define Resource Interactions

A resource's model provides its interactions, what it can do and what can be done to it.

Using an HTTP method and the resource's URI can provide the bulk of semantics. The media type provides a representation or desired change to the representation. In some cases, such as HTTP's POST method, the client can request an interaction to occur according to the media type's semantics.

Sufficiently complex resource may have sub-resources to enable addressable interactions via POST for whens HTTPs methods and POST toward the resource itself are inadequate.

Common Resource Archetypes

Resource archtypes help guide common interactions and semantics for common resource types. These may help inform, but do not constrain, modeling for resources or an API.

Instance	An individual thing; often a document
Collection	A server-managed group of like things
Store	A client-managed group of things
Controller	A sub-resource providing a method

URI and Resource Plan: Typical REST Semantics

URI Template	Archetype	Representation	GET	PUT	POST	DELETE	HEAD	OPTIONS
/cars	collection	`application/json` A CarsCollection object	Get representation of a Collection		Create Instance OR Process	Delete the collection	Get representation metadata	Get interaction options
/cars/{car-id}	instance	`application/json` A Car object	Get representation of a car	Update a car's representation	Process by Resource Semantics	Delete car from the collection	Get representation metadata	Get interaction options
/cars/{car-id}/honk	controller	No representation			Honk: Process by Resource Semantics		Get representation metadata	Get interaction options

Hypermedia: Application State Managed through Links

IANA Link Relations

Linking via hypermedia is core constraint of REST. The client and developer should not need a reference to the web service's functionality, since the service provides sufficient information about the resource and opportunities for interaction via representations and links. From the perspective of REST, Web APIs are not different from the Web viewed experienced through browsers.

Early on, structured object formats had no way to support linking the way HTML did. As time has gone on, numerous methods have emerged.

HTTP Link Header

```
Link: <https://restful.ws/posters/reference>; rel="canonical"; title="REST-
ful Web Services and HTTP: Standards and Sources"; type=application/pdf
```

JSON Hyper-Schema uses an external schema a client may use to validate the payload and understand its links. With Hyper-Schema, the client can self-direct subsequent requests and provide end user navigation.

JSON Representation

```
{
  "id": 1,
  "sandwich": "BLT"
}
```

JSON Representation

```
{
  "id": 442,
  "sandwich": "PB&J"
}
```

JSON Schema and Hyper-Schema

```
{
  "properties": {
    "id": { "type": "integer" },
    "sandwich": { "type": "string" }
  },
  "links": [{
    "rel": "self",
    "href": "/sandwiches/{sandwich-id}",
    "templatePointers": {
      "sandwich-id": "/id"
    }
}]}}
```

Per Web Linking (rfc8288), a link is a connection between two resources comprised of:

- a **link context** - the representation providing the link
- a **link relation type** - the nature of the link relation
- a **link target**
- optional **target attributes**
  - **type** - expected media type for
  - **hreflang** - a language tag-
  - **media** - a CSS @media value
  - **title** - A human-readible label for the destination
  - **title*** - Title with specified character set
  - *extended target attributes* are allowed

- **Linking for Web Pages**: Web pages can use use anchor hrefs, image links, and full-fledged Link header tags.
- **Linking via Header**: The *Web Linking* (rfc8288) standard provides a rich link description format thtat may be used in the Link header. This makes linking possible for formats, including pictures, abitrary files, and structured formats that do not have support for the feature.
- **Linking via JSON Payload**: JSON Hyper-Schema, part of in-progress JSON Schema work, will formalize links in JSON aligned to IETF's Web Linking.
- OpenAPI 3 supports links semantically. Presenting the links in headers and representations is up to the API designer.