Verbose Documentation

Release 0.4.0

Stephen Mizell

September 17, 2014

Contents

Verbose is a general-purpose, multi-use hypermedia format that can be used to build robust and highly-descriptive APIs.

Note: Verbose is currently under development, which means things could change before it's ready to be used. Please keep this in mind if you desire to start using it.

CHAPTER 1

Content

1.1 Overview

1.1.1 Description

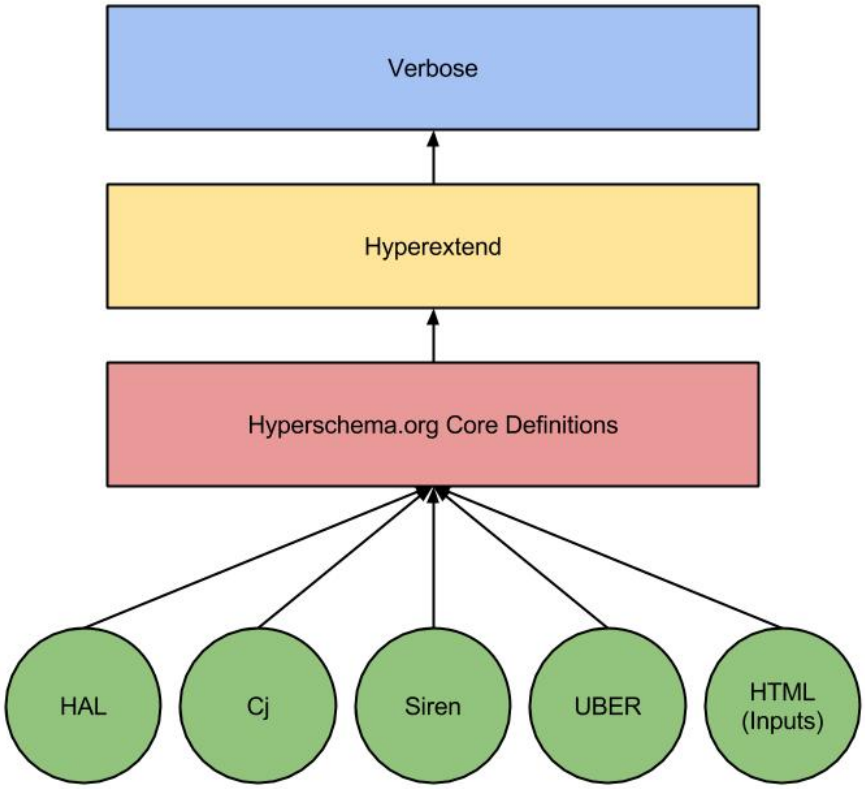
Verbose is a general-purpose, multi-use hypermedia format that can be used to build robust and highly-descriptive APIs. Because it aims to cover as many scenarios as possible, it can be used in many situations, such as:

- Resource representation
- Link Relation
- Profile
- Documentation
- Object Model

It can be as minimal or verbose as necessary to fit the situation. Because of it's design, it should be able to cover almost anything that other hypermedia formats can cover.

1.1.2 Design

Verbose comes from a need to have an object that can handle aspects of many of the popular hypermedia formats. To do this, these other hypermedia formats were taken down to their bare essentials. These essentials make up the Hypermedia Core Definitions.



From these core definitions, the Hyperextend library was created. Hyperextend is a library of components that can be used to extend any other format to add in hypermedia elements (e.g. adding form-like semantics to HAL).

Verbose was then created by combining all of these components into one format. This essentially makes it a format that is derived from all of the media types found on the Hyperschema.org site.

1.2 Status

Current Version: 0.4

Current Release: 0.4.0

Build Date: September 17, 2014

1.3 Specification

Current Version: 0.4

Current Release: 0.4.0

Build Date: September 17, 2014

Proposed Registration: `application/vnd.verbose+json`

The document outlines the Verbose hypermedia format.

Note: This is currently in active development. There may be parts of this change before it is ready for use.

Note: The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1.3.1 Definitions

Since Verbose reuses a lot of properites, this section will first define some of these common properties.

Hypermedia

These properties are used to define hypermedia information and hints related to links, queries, actions, and resources.

`embedAs` A way to tell the client how an item should be embedded. These are the possible values:

1. image
2. audio
3. video
4. text
5. application

If left blank, it is assumed to be a normal link.

`rels` An `array` of link relations for an item.

`responseTypes` An `array` of available media types on the server.

Types

`typesOf` A way to specify what type of item an item is. Useful when specifying Schema.org values or linking to profiles. It is an array of strings. If multiple types are specified, the first type is of most priority and the last type is of least, in the event that there are conflicting restraints. Having several types is an option, but it may be better to create and define a new type if there are too many.

By default, Verbose supports the JSON primitives:

1. string
2. number
3. boolean
4. array
5. object

You can of course create your own types and define certain constraints and validations on the object. See the

Extensibility

Each of these properties use *Verbose Path* to reference items from the same document or other documents. Please see the documentation on this to understand how Verbose Path works.

extend A way to extend the item with other items. It is an object, the contents of which can be defined outside of this spec.

forEach An `array` of Verbose Paths that specify for what item a template can be used. These templates can be for links, queries, actions, or resource templates.

If the Verbose Path string specified references more than one item in the document, it means that template can be used for each of those items.

```
{
  "verbose": {
    "href": "/customers",
    "templates": [
      {
        "forEach": [ "#", "#/includes[rel=item]" ]
      }
    ]
  }
}
```

In this example above, the `forEach` array contains a reference to `#`, which references the root resource of the document, and `#/includes[rel=item]`, which references any included items with a link relation of `item`. Please see the Verbose Path section to see how it is used.

mapsTo An `array` of Verbose Path strings (see section for details on how this is used)

1.3.2 Resource

A Verbose Resource is an `object` for defining everything dealing with a particular resource. It uses these properties from the definition list.

1. `id` - Unique identifier for resource
2. `name` - Name of resource
3. `typesOf` - For pointing to another semantic or schema for the resource

It also supports.

id Unique identifier for resource

name Name of resource

typesOf For pointing to another semantic or schema for the resource

meta A *Meta object*

href Link to the resource

prefixes An `array` of *Prefix objects*

semantics An `array` of *Semantic objects*

properties A *Properties object*

links An `array` of *Link objects*

queries An `array` of *Query objects*

actions An `array` of *Action objects*

templatedLinks An `array` of *Templated Link objects*

templatedActions An `array` of *Templated Action objects*

templates An `array` of *Resource Template objects*

includes An `array` of full *Resource objects*

errors An *Error object*

See the *Examples* page for examples of a resource

1.3.3 Namespace

All Verbose documents MUST have a `verbose` namespace.

```
{
  "verbose": {}
}
```

1.3.4 Prefixes

Prefixes can be used to shorten URLs. When used, they are available throughout the entire document.

prefixes This is an `array` of prefix objects.

- **prefix** The short prefix name.
- **href** The URL to be used as the prefix.

Example

```
{
  "verbose": {
    "version": "0.4",
    "prefixes": [
      {
        "prefix": "schema",
        "href": "http://schema.org/"
      }
    ]
  }
}
```

1.3.5 Properties

The `properties` object is simply a JSON object. Its semantics are defined by the Semantic object.

1.3.6 Semantics

The `semantics` array is an array of Semantic objects. It supports the following properites listed in the *Definitions* list:

1. `id` - Unique identifier for semantic

2. `name` - Name of property being defined
3. `title` - Title of semantic
4. `description` - Description of semantic
5. `label` - Human-readable label or prompt for semantic
6. `typesOf` - For pointing to another semantic or schema for the property
7. `mapsTo` - Property to which the semantic point

Example

Below is an example showing a resource that has `properties` and `semantics` for those properties. In this example, the property is `email`, which is a `string` and uses the HTML5 formatting for `email`. The instance data for that property is `john@doe.com`.

```
{
  "verbose": {
    "semantics": [
      {
        "name": "email",
        "type": "string",
        "format": "email",
        "label": "Email",
        "mapsTo": "#/properties.email"
      }
    ],
    "properties": {
      "email": "john@doe.com"
    }
  }
}
```

1.3.7 Field

A Field object supports the following properites listed in the *Definitions* list:

1. `id` - Unique identifier for field
2. `name` - Name of field
3. `title` - Title of field
4. `description` - Description of field
5. `label` - Human-readable label or prompt for field
6. `typesOf` - Types of the field
7. `extend` - Added details determined by the type

A `field` object also provides the following properties:

`defaultValue` The optional default value of the field. This is a `string`.

`currentValue` The current value of the field. This is a `string`.

`value` The value of the field which cannot be changed. The `defaultValue` and `currentValue` properties allow for the values to be changed or set, though the `value` property is unchangeable. It is a way for the API to provide unchangeable field data, equivalent to a hidden field in HTML.

`options` An `array` of option objects. Option objects have a `name` and `value` property for each option.

1.3.8 Transitions

A transition is an available progression from one state to another state. There are many characteristics of transitions, and several different categories that hypermedia formats use. Transitions have been broken down into links, queries, actions, templated links, templated actions, and embedded resources below.

Links

The `links` property is an array of Link objects. It supports all of the properties of the *Resource object*, along with the following properites listed in the *Definitions* list:

1. `label` - Human-readable label or prompt for link
2. `rels` - Link relations of the link
3. `responseTypes` - Types with which the server may respond
4. `embedAs` - Ways to inform the client how an item should be transcluded
5. `typesOf` - For pointing to another semantic or schema for the link

A link is always considered safe. The HTTP method `GET` should be used for these requests.

Example

The link below provides a link to a customer resource.

- It shows `name` being used, which has a name of `customer`
- It defines the link relations for this link using the `rels` property
- It uses `responseTypes` to hint at what representations are available from the server
- It uses `href` to provide the actual URL to the resource

```
{
  "verbose": {
    "links": [
      {
        "name": "customer",
        "rels": [ "item", "http://example.com/rels/customer"],
        "responseTypes": [
          "application/json",
          "application/hal"
        ],
        "href": "/customer/4"
      }
    ]
  }
}
```

Queries

Queries are safe GET requests that provide a way for specifying query parameters.

The queries property is an array of Query objects. It supports all of the properties of the *Resource object*, along with the following properites listed in the *Definitions* list:

1. `label` - Human-readable label or prompt for link
2. `rels` - Link relations of the link
3. `responseTypes` - Types with which the server may respond
4. `embedAs` - Ways to inform the client how an item should be transcluded
5. `typesOf` - For pointing to another semantic or schema for the link
6. `queryParams` - An `array` of `field` objects to be used for query parameters

Example

This example shows how `queryParams` are used.

```
{
  "verbose": {
    "queries": [
      {
        "name": "customer",
        "rels": [ "search"],
        "responseTypes": [
          "application/json",
          "application/hal"
        ],
        "href": "/customers",
        "queryParams": [
          {
            "name": "email",
            "label": "Email"
          }
        ]
      }
    ]
  }
}
```

Actions

An action is a way to provide an unsafe action.

The `actions` property is an array of Action objects. It supports the following properites listed in the *Definitions* list:

1. `id` - Unique identifier for item
2. `name` - Name of action
3. `title` - Title of action
4. `description` - Description of action
5. `label` - Human-readable label or prompt for action
6. `rels` - Link relation of the action
7. `responseTypes` - Types with which the server may respond
8. `requestTypes` - Types in which the server accepts

9. `embedAs` - Ways to inform the client how an item should be transcluded
10. `href` - URL for the action
11. `typesOf` - For pointing to another semantic or schema for the action
12. `method` - HTTP method for the action
13. `bodyParams` - An `array` of `field` objects that is used for specifying the parameters for the body of a request

The HTTP method for the action MUST be `POST`, `PUT`, or `DELETE`.

Example

This action can be used to create a customer.

- It uses the `POST` method
- It has two body parameters: `first_name` and `last_name` which are both strings

```
{
  "verbose": {
    "actions": [
      {
        "title": "Add Customer",
        "rels": [ "append"],
        "href": "/customers",
        "method": "POST",
        "bodyParams": [
          {
            "name": "first_name",
            "label": "First Name"
          },
          {
            "name": "last_name",
            "label": "Last Name"
          }
        ]
      }
    ]
  }
}
```

Templated Links

The `templatedLinks` property is an array of Templated Link objects. It supports the following properites listed in the *Definitions* list:

1. `id` - Unique identifier for item
2. `name` - Name of link
3. `title` - Title of link
4. `description` - Description of link
5. `label` - Human-readable label or prompt for link
6. `rels` - Link relations of the link

7. `responseTypes` - Types with which the server may respond
8. `embedAs` - Ways to inform the client how an item should be transcluded
9. `hreft` - URI template for the link
10. `typesOf` - For pointing to another semantic or schema for the link
11. `uriParams` - An `array` of `field` objects to be used for URI template parameters

Example

This shows a resource that has a templated link for a customer resource This is very similar to a regular link, but it provides a `hreft` property, which is a templated URL, along with URI parameters.

In this case, there is one URI parameters call `id`, which is a number.

```
{
  "verbose": {
    "templatedLinks": [
      {
        "name": "customer",
        "rels": [ "item", "http://example.com/rels/customer"],
        "responseTypes": [
          "application/json",
          "application/hal"
        ],
        "hreft": "/customer/{id}",
        "uriParams": [
          {
            "name": "id"
          }
        ],
      }
    ]
  }
}
```

Templated Actions

The `templatedActions` property is an array of Templated Action objects. It supports the following properites listed in the *Definitions* list:

1. `id` - Unique identifier for item
2. `name` - Name of action
3. `title` - Title of action
4. `description` - Description of action
5. `label` - Human-readable label or prompt for action
6. `rels` - Link relation of the action
7. `responseTypes` - Types with which the server may respond
8. `requestTypes` - Types in which the server accepts
9. `embedAs` - Ways to inform the client how an item should be transcluded
10. `hreft` - URI template for the action

11. `typesOf` - For pointing to another semantic or schema for the action
12. `method` - HTTP method for the action
13. `bodyParams` - An `array` of `field` objects that is used for specifying the parameters for the body of a request
14. `uriParams` - An `array` of `field` objects that is used for specifying the parameters for the URI template

Example

This templated action provides an action for editing any customer. This allows for including actions that can be used for multiple resources without including the action multiple times.

In this example, there are both URI parameters and body parameters for building the request.

```
{
  "verbose": {
    "templatedActions": [
      {
        "title": "Edit Customer",
        "rels": [ "http://example.com/rels/customer"],
        "hreft": "/customer/{id}",
        "method": "PUT",
        "uriParams": [
          {
            "name": "id"
          }
        ],
        "bodyParams": [
          {
            "name": "first_name",
            "label": "First Name"
          },
          {
            "name": "last_name",
            "label": "Last Name"
          }
        ]
      }
    ]
  }
}
```

Embedded Resources

Included resources are just to be considered as included resources and MAY be full representations. The reason for this and the `partials` property is that it allows for explicitly telling the client that the resource needs to be requested if the full resource is desired.

1.3.9 Resource Template

This item uses the `forEach` from the *Definitions* list. It also supports all of the properties that a transition supports except for the `href` and `hreft` properties.

Example

This is an example of a resource that provides templates for working with embedded resources. It shows this template can be used for any included resource with `item` as a rel, and uses `forEach` to specify this.

```
{
  "verbose": {
    "href": "/customers",
    "templates": [
      {
        "rels": [ "update" ],
        "forEach": [ "#", "#/includes[rel=item]" ],
        "requestTypes": [ "application/x-www-form-urlencoded" ],
        "method": "PUT"
        "fields": [
          {
            "name": "first_name",
            "type": "string",
            "label": "First Name"
          },
          {
            "name": "last_name",
            "type": "string",
            "label": "Last Name"
          }
        ]
      }
    ],
    "includes": [
      {
        "rels": [ "item" ],
        "href": "/customers/1",
        "properties": {
          "first_name": "John",
          "last_name": "Doe"
        }
      },
      {
        "rels": [ "item" ],
        "href": "/customers/2",
        "properties": {
          "first_name": "Jane",
          "last_name": "Doe"
        }
      }
    ]
  }
}
```

1.3.10 Meta

The `meta` property is a resource object that can be used to specify meta data, such as meta links or properties. The properties and links for the error are left up to the designer.

```
{
  "versbose": {
    "version": "0.4",
```

```
    "meta": {
      "transitions": [
        {
          "rels": [ "self" ],
          "href": "/customers"
        }
      ]
    }
  }
}
```

1.3.11 Errors

The `errors` property is a resource object that can be used specifically for errors. The properties and links for the error are left up to the designer.

```
{
  "versbose": {
    "version": "0.4",
    "errors": {
      "properties": {
        "message": "There was an error when creating this resource"
      }
    }
  }
}
```

1.3.12 Verbose Path

Verbose Path is a way to reference objects throughout a Verbose document or in other Verbose documents. It is very simple and tries to only provide what is needed to reference items throughout a document.

Root Resource

The # alone SHOULD be considered the path to the root resource of a Verbose document. The example below shows a template that can be used for the root resource.

```
{
  "verbose": {
    "version": "0.4",
    "templates": [
      {
        "forEach": [ "#" ],
        "method": "POST",
        "rels": [ "create" ],
        "fields": [
          { "name": "first_name" },
          { "name": "last_name" }
        ]
      }
    ]
  }
}
```

ID

This shows the template can be used for the item where the ID is equal to `person`.

```
{
  "verbose": {
    "version": "0.4",
    "templates": [
      {
        "forEach": [ "#person" ],
        "method": "POST",
        "rels": [ "edit" ],
        "fields": [
          { "name": "first_name" },
          { "name": "last_name" }
        ]
      }
    ],
    "includes": [
      {
        "id": "person",
      }
    ]
  }
}
```

Nested Properties

Properties of an object can be specified with a dot. Shown below, the semantics `fullName` and `email` are mapped to properties of the `customer` object.

```
{
  "verbose": {
    "version": "0.4",
    "semantics": [
      {
        "name": "customer",
        "type": "object",
        "mapsTo": "#/properties.customer"
      },
      {
        "name": "fullName",
        "type": "string",
        "mapsTo": "#/properties.customer.fullName"
      },
      {
        "name": "email",
        "type": "string",
        "mapsTo": "#/properties.customer.email"
      }
    ],
    "properties": {
      "customer": {
        "fullName": "John Doe",
        "email": "johndoe@example.com"
      }
    }
```

```
  }
}
```

Arrays

Arrays can also be referenced.

```
{
  "verbose": {
    "version": "0.4",
    "semantics": [
      {
        "name": "customers",
        "type": "array",
        "mapsTo": "#/properties.customers[]"
      },
      {
        "name": "fullName",
        "type": "string",
        "mapsTo": "#/properties.customers[].fullName"
      },
      {
        "name": "email",
        "type": "string",
        "mapsTo": "#/properties.customers[].email"
      }
    ],
    "properties": {
      "customers": [
        {
          "fullName": "John Doe",
          "email": "johndoe@example.com"
        },
        {
          "fullName": "Jane Doe",
          "email": "janedoe@example.com"
        }
      ]
    }
  }
}
```

Filtering Arrays

The square brackets can be used to filter arrays. The example below shows the template is usable for all included resources with the name equal to customer.

```
{
  "verbose": {
    "version": "0.4",
    "templates": [
      {
        "forEach": [ "#/includes[name=customer]" ],
        "method": "PUT",
        "rels": [ "update" ],
        "fields": [
```

```
          { "name": "first_name" },
          { "name": "last_name" }
        ]
      }
    ],
    "includes": [
      {
        "name": "customer",
        "properties": {
          "customer": {
            "fullName": "John Doe",
            "email": "johndoe@example.com"
          }
        }
      }
    ]
  }
}
```

1.4 Examples

1.4.1 Compared to Other Formats

Because of *Verbose's Design*, it can handle just about anything other formats can handle.

HAL+JSON

This example is the example from the HAL spec. Because the HAL spec says you should not assume the embedded resources are full resources, I've put them in the *partials* array.

```
{
  "verbose": {
    "version": "0.4",
    "prefixes": [
      {
        "prefix": "ea",
        "href": "http://example.com/docs/rels/"
      }
    ],
    "properties": {
      "currentlyProcessing": 14,
      "shippedToday": 20
    },
    "links": [
      {
        "rels": [ "self" ],
        "href": "/orders"
      },
      {
        "rels": [ "next" ],
        "href": "/orders?page=2"
      },
      {
        "rels": [ "ea:admin" ],
```

```
        "href": "/admins/2",
        "label": "Fred"
      },
      {
        "rels": [ "ea:admin" ],
        "href": "/admins/5",
        "label": "Kate"
      }
    ],
    "includes": [
      {
        "rels": [ "ea:order" ],
        "properties": {
          "total": 30.00,
          "currency": "USD",
          "status": "shipped"
        },
        "links": [
          {
            "rels": [ "self" ],
            "href": "/orders/123"
          },
          {
            "rels": [ "ea:basket" ],
            "href": "/baskets/98712"
          },
          {
            "rels": [ "ea:customer" ],
            "href": "/customers/7809"
          }
        ]
      },
      {
        "rels": [ "ea:order" ],
        "properties": {
          "total": 20.00,
          "currency": "USD",
          "status": "processing"
        },
        "links": [
          {
            "rels": [ "self" ],
            "href": "/orders/124"
          },
          {
            "rels": [ "ea:basket" ],
            "href": "/baskets/98713"
          },
          {
            "rels": [ "ea:customer" ],
            "href": "/customers/12369"
          }
        ]
      }
    ]
  }
}
```

Siren

This is taken from the example in the Siren spec. I took a little bit of liberty with this one and considered one of the embedded entities to be a partial representation.

```
{
  "verbose": {
    "typesOf": [ "order" ],
    "properties": {
        "orderNumber": 42,
        "itemCount": 3,
        "status": "pending"
    },
    "links": [
      {
        "rels": [ "self" ],
        "href": "http://api.x.io/orders/42"
      },
      {
        "rels": [ "previous" ],
        "href": "http://api.x.io/orders/41"
      },
      {
        "rels": [ "next" ],
        "href": "http://api.x.io/orders/43"
      }
    ],
    "actions": [
      {
        "name": "add-item",
        "title": "Add Item",
        "method": "POST",
        "href": "http://api.x.io/orders/42/items",
        "requestTypes": [ "application/x-www-form-urlencoded" ],
        "bodyParams": [
          {
            "name": "orderNumber",
            "typesOf": [ "number" ],
            "value": "42"
          },
          {
            "name": "productCode",
            "typesOf": [ "string" ],
          },
          {
            "name": "quantity",
            "typesOf": [ "number" ]
          }
        ]
      },
    ],
    "includes": [
      {
        "typesOf": [ "items", "collection" ],
        "rels": [ "http://x.io/rels/order-items" ],
        "href": "http://api.x.io/orders/42/items"
      },
      {
        "typesOf": [ "info", "customer" ],
```

```
        "rels": [ "http://x.io/rels/customer" ],
        "properties": {
          "customerId": "pj123",
          "name": "Peter Joseph"
        },
        "links": [
          {
            "rels": [ "self" ],
            "href": "http://api.x.io/customers/pj123"
          }
        ]
      }
    ]
  }
}
```

Collection+JSON

```
{
  "verbose" : {
    "version" : "0.4",
    "rels": [ "collection" ],
    "href" : "http://example.org/friends/",
    "links" : [
      {
        "rels" : [ "feed" ],
        "href" : "http://example.org/friends/rss"
      }
    ],
    "queries": [
      {
        "rels" : [ "search" ],
        "href" : "http://example.org/friends/search",
        "label" : "Search",
        "queryParams" : [
          {
            "name" : "search",
            "defaultValue" : ""
          }
        ]
      }
    ],
    "includes" : [
      {
        "rels": [ "item" ],
        "href" : "http://example.org/friends/jdoe",
        "semantics" : [
          {"name" : "full-name", "label" : "Full Name"},
          {"name" : "email", "label" : "Email"}
        ],
        "properties": {
          "full-name": "J. Doe",
          "email": "jdoe@example.org"
        },
        "transitions" : [
          {
            "rels" : [ "blog" ],
```

```
        "href" : "http://examples.org/blogs/jdoe",
        "label" : "Blog"
      },
      {
        "rels" : [ "avatar" ],
        "href" : "http://examples.org/images/jdoe",
        "label" : "Avatar",
        "embedAs" : "image"
      }
    ]
  },
  {
    "rels": [ "item" ],
    "href" : "http://example.org/friends/msmith",
    "semantics" : [
      {"name" : "full-name", "label" : "Full Name"},
      {"name" : "email", "label" : "Email"}
    ],
    "properties": {
      "full-name": "M. Smith",
      "email": "msmith@example.org"
    },
    "transitions" : [
      {
        "rels" : [ "blog" ],
        "href" : "http://examples.org/blogs/msmith",
        "label" : "Blog"
      },
      {
        "rels" : [ "avatar" ],
        "href" : "http://examples.org/images/msmith",
        "label" : "Avatar",
        "embedAs" : "image"
      }
    ]
  },
  {
    "rels": [ "item" ],
    "href" : "http://example.org/friends/rwilliams",
    "semantics" : [
      {"name" : "full-name", "label" : "Full Name"},
      {"name" : "email", "label" : "Email"}
    ],
    "properties": {
      "full-name": "R. Williams",
      "email": "rwilliams@example.org"
    },
    "transitions" : [
      {
        "rels" : [ "blog" ],
        "href" : "http://examples.org/blogs/rwilliams",
        "label" : "Blog"
      },
      {
        "rels" : [ "avatar" ],
        "href" : "http://examples.org/images/rwilliams",
        "label" : "Avatar",
        "embedAs" : "image"
```

```
          }
        ]
      }
    ],
    "templates" : [
      {
        "forEach": [ "#" ],
        "rels": [ "append" ],
        "method": "POST",
        "fields" : [
          { "name" : "full-name", "defaultValue" : "", "label" : "Full Name" },
          { "name" : "email", "defaultValue" : "", "label" : "Email" },
          { "name" : "blog", "defaultValue" : "", "label" : "Blog" },
          { "name" : "avatar", "defaultValue" : "", "label" : "Avatar" }
        ]
      },
      {
        "forEach": [ "#/includes[rel=item]" ],
        "rels": [ "update" ],
        "method": "PUT",
        "fields" : [
          { "name" : "full-name", "defaultValue" : "", "label" : "Full Name" },
          { "name" : "email", "defaultValue" : "", "label" : "Email" },
          { "name" : "blog", "defaultValue" : "", "label" : "Blog" },
          { "name" : "avatar", "defaultValue" : "", "label" : "Avatar" }
        ]
      }
    ]
  }
}
```

JSON API

This takes the example from the JSON API page. There are several ways to do this in Verbose, so below are a couple of different examples.

This example lets the templated links map its parameters to specific properties in the document.

```
{
  "verbose": {
    "version": "0.4",
    "properties": {
      "id": 1,
      "title": "Rails is Omakase",
      "author_id": "9",
      "comment_ids": [ "5", "12", "17", "20" ]
    },
    "templatedLinks": [
      {
        "typesOf": [ "author", "people" ],
        "hreft": "http://example.com/people/{author_id}",
        "uriParams": [
          {
            "name": "author_id",
            "mapsTo": [ "#/properties.author_id" ]
          }
        ]
```

```
      },
      {
        "typesOf": [ "comments" ],
        "hreft": "http://example.com/comments/{comment_id}",
        "uriParams": [
          {
            "name": "comment_id",
            "mapsTo": [ "#/properties.comment_ids[]" ]
          }
        ]
      }
    ]
  }
}
```

1.4.2 Link Relation Example

Link Relation

```
{
  "verbose": {
    "version": "0.4",
    "href": "http://example.com/rels/customers",
    "title": "Customer Collection",
    "description": "This is a collection of customers",
    "typesOf": [ "customers" ],
    "semantics": [
      {
        "title": "Total Customers",
        "description": "The total number of customers in this collection.",
        "name": "total_customers",
        "typesOf": [ "number" ]
      }
    ],
    "actions": [
      {
        "title": "Append Customer",
        "description": "Action for adding a new customer",
        "method": "POST",
        "requestTypes": [ "application/x-www-form-urlencoded" ],
        "fields": [
          { "name": "first_name", "typesOf": [ "string" ], "label": "First Name" },
          { "name": "last_name", "typesOf": [ "string" ], "label": "Last Name" }
        ]
      }
    ],
    "includes": [
      {
        "id": "customer",
        "typesOf": [ "customer" ],
        "rels": [ "item" ],
        "semantics": [
          {
            "title": "First Name",
            "description": "First name of customer",
            "name": "first_name",
```

```
            "typesOf": [ "string" ]
          },
          {
            "title": "Last Name",
            "description": "Last name of customer",
            "name": "last_name",
            "types": [ "string" ]
          }
        ],
        "actions": [
          {
            "title": "Update Customer",
            "description": "Action for adding a new",
            "method": "PUT",
            "requestTypes": [ "application/x-www-form-urlencoded" ],
            "fields": [
              { "name": "first_name", "typesOf": [ "string" ], "label": "First Name" },
              { "name": "last_name", "typesOf": [ "string" ], "label": "Last Name" }
            ]
          }
        ]
      }
    ]
  }
}
```

Resource Representation

```
{
  "verbose": {
    "version": "0.4",
    "typesOf": [ "customers" ],
    "rels": [ "http://example.com/rels/customers" ],
    "href": "/customers",
    "properties": {
      "total_customers": 45
    },
    "includes": [
      {
        "rels": [ "http://example.com/rels/customers#customer" ],
        "typesOf": [ "customer" ],
        "href": "/customers/1",
        "rels": [ "item" ],
        "properties": {
          "first_name": "John",
          "last_name": "Doe"
        }
      },
      {
        "rels": [ "http://example.com/rels/customers#customer" ],
        "typesOf": [ "customer" ],
        "href": "/customers/2",
        "rels": [ "item" ],
        "properties": {
          "first_name": "Jane",
          "last_name": "Doe"
        }
```

```
      }
    ]
  }
}
```

1.4.3 Profile

Profile

```
{
  "verbose": {
    "version": "0.4",
    "title": "Collection of Customers",
    "description": "A collection of customers",
    "id": "customers",
    "rels": [ "collection" ],
    "queries": [
      {
        "id": "search",
        "rels": [ "search" ],
        "description": "Customer search",
        "queryParams": [
          {
            "title": "Company Name",
            "description": "Company name search field",
            "name": "companyName"
          },
          {
            "title": "Email Address",
            "description": "Email address search field",
            "name": "email"
          }
        ]
      }
    ],
    "includes": [
      {
        "id": "customer",
        "rels": [ "item" ],
        "description": "Customer resource",
        "semantics": [
          { "name": "companyName" },
          { "name": "firstName" },
          { "name": "lastName" },
          { "name": "email" },
          { "name": "phone" }
        ]
      }
    ]
  }
}
```

Resource Representation


```
{
  "verbose": {
    "version": "0.4",
    "id": "customers",
    "rels": [ "collection" ],
    "typeOf": "http://example.com/customers#customers",
    "links": [
      {
        "rels": [ "profile" ],
        "href": "http://example.com/customers"
      }
    ],
    "queries": [
      {
        "id": "search",
        "rels": [ "search" ],
        "typeOf": "http://example.com/customers#search",
        "description": "Customer search",
        "queryParams": [
          {
            "title": "Company Name",
            "name": "companyName"
          },
          {
            "title": "Email Address",
            "name": "email"
          }
        ]
      }
    ],
    "includes": [
      {
        "typesOf": [ "customer" ],
        "href": "/customer/1",
        "rels": [ "item" ],
        "typeOf": "http://example.com/customers#customer",
        "properties": {
          "companyName": "ACME, Inc.",
          "firstName": "John",
          "lastName": "Doe",
          "email": "john@acme.com"
        }
      },
      {
        "typesOf": [ "customer" ],
        "href": "/customer/2",
        "rels": [ "item" ],
        "typeOf": "http://example.com/customers#customer",
        "properties": {
          "companyName": "ACME, Inc.",
          "firstName": "Jane",
          "lastName": "Doe",
          "email": "jane@acme.com"
        }
      }
    ]
```

```
  }
}
```

1.5 Schema

> **Warning:** This schema is out of date. Do not use. Will be updated soon.

Provided here is a JSON schema for validating Verbose documents.:

```
{
  "properties":{
    "verbose": { "$ref": "#/definitions/verbose" }
  },
  "definitions": {
    "verbose": {
      "allOf": [
        { "$ref": "http://hyperschema.org/extensions/hyperextend/link#" },
        { "$ref": "#/definitions/resource" }
      ]
    },
    "resource": {
      "properties": {
        "semantics": {
          "type": "array",
          "items": { "$ref": "http://hyperschema.org/extensions/hyperextend/semantic#" }
        },
        "properties": {
          "type": "object"
        },
        "affordances": {
          "type": "array",
          "items": { "$ref": "http://hyperschema.org/extensions/hyperextend/affordance#" }
        },
        "templates": {
          "type": "array",
          "items": { "$ref": "http://hyperschema.org/extensions/hyperextend/resourcetemplate#" }
        },
        "partials": {
          "type": "array",
          "items": { "$ref": "#/definitions/resource" }
        },
        "includes": {
          "type": "array",
          "items": { "$ref": "#/definitions/resource" }
        },
        "errors": { "$ref": "#/definitions/resource" }
      }
    }
  }
}
```

CHAPTER 2

Issues, Questions, Comments

If you find any issues, have any questions, or would like to comment on any aspect of the documentation or format design, please so on the Github issue page.

CHAPTER 3

Contributing

The source for this documentation can be found on Github at https://github.com/smizell/verbose. To contribute, please open a pull request. Thanks for checking out Verbose!