RESTful Service Description Language (RSDL)

Describing RESTful Services Without Tight Coupling

Jonathan Robie, EMC Corporation <jonathan.robie@emc.com>

Rob Cavicchio, EMC Corporation <rob.cavicchio@emc.com>

Rémon Sinnema, EMC Corporation <remon.sinnema@emc.com>

Erik Wilde, EMC Corporation <erik.wilde@emc.com>

Abstract

RESTful Service Description Language (RSDL) is an XML vocabulary for designing and documenting hypermedia-driven RESTful Services. RSDL takes a purist hypermedia-driven approach to REST design, requiring that a service have a single entry point, and focusing the design on resources, links, and media types.

Table of Contents

Introduction

RESTful Service Description Language (RSDL) is an XML vocabulary for designing and documenting hypermedia-driven RESTful Services. RSDL takes a purist hypermedia-driven approach to REST design, requiring that a service have a single entry point, and focusing the design on resources, links, and media types. By representing the concepts of REST in a formal vocabulary, RSDL helps designers of an interface think more clearly about the design process. In our work at EMC Corporation, we teach and coach RESTful design, evaluate proposed REST interfaces for products and internal services, and produce documentation for these interfaces. We created RSDL to make our work more efficient, and we have begun using it as a teaching tool, a standard format for specifications that allows

us to track design changes in source code control, test for design consistency with specification-driven programs, and produce documentation.

The REST community has little consensus on how to document a RESTful service beyond this well-known Roy Fielding quote:

> Any effort spent describing what methods to use on what URIs of interest should be entirely defined within the scope of the processing rules for a media type (and, in most cases, already defined by existing media types). [Failure here implies that out-of-band information is driving interaction instead of hypertext.]

While this gives excellent guidance about things that should not be documented, it tells us that we should focus on documenting a media type without telling us how to do so. Well-designed RESTful services are loosely coupled, allowing a client to use the service with no prior knowledge beyond an initial URI and a set of media types. But this gives little concrete guidance to a person who is specifying a RESTful service, and needs to know precisely what information should be provided to a client, which fundamental concepts should be documented for a media type, how dependencies on multiple media types in a single service should be represented, where to document semantics specific to the application domain, or how to organize all of the information that belongs in a specification.

This paper presents RESTful Service Description Language (RSDL, pronounced "risdle"), an XML vocabulary that provides a structured way to specify a RESTful service. RSDL is still being developed.

What should a RESTful service description describe?

The description for a RESTful service should describe semantics specific to the service that go beyond functionality known to a generic REST client. In RSDL, every RESTful service has a single entry point, which corresponds to a home resource, and all other resources can be discovered from the home resource using links. The structure of a RSDL description enforces these design constraints.

A RSDL description focuses on describing the following items:

- Media types, with documentation and an optional link to a schema or description that documents each media type used in a service. This includes the kinds of response bodies sent by the server, and how to find links in them and identify specific kinds of link.
- Resources, designating one resource as the service entry point.
- Links for each resource and the resources they refer to.
- Methods allowed for each resource, and the associated requests and responses.
- HTTP headers, including custom headers.
- Authentication mechanisms and identity providers, which can be specified for the entire service or at the individual resource level.
- URI parameters and URI templates.
- HTTP status codes, which are described at the service level (not at the individual request level or the resource level), but can be referenced at the request level if need be.

It is important to clearly understand the relationship between services, resources, links, URIs, methods, representations, and media types. A RESTful service is a collection of resources, which are identified by URIs. To start using a service, a client needs an entry point. A service's published entry point is the URI of the home resource. All other resources should be discoverable from the home resource using links, which identify URIs that refer to other resources, and can be used for HTTP requests. A client can do a `GET` to obtain a representation of a resource in, say, XML or JSON. A representation is a sequence of bytes that represents the current state of a resource; the syntax and semantics of a representation are defined by the corresponding media type. Note that a representation is not the same thing as a resource,

it is merely the data required to represent the resource's state. You cannot send HTTP requests to a representation, you send HTTP requests to a resource. You cannot parse or generate a resource, you can parse or generate a representation. These distinctions are important, and they are reflected in the structure of RSDL. For instance, a media type can provide schemas or descriptions that describe the structure of its representations. A resource describes the HTTP requests that can be applied to a resource. In RSDL, we view links as an abstraction that represents the potential relationship among resources, they have a concrete representation in a given media type. If a given resource has both an XML representation and a JSON representation, the same links will be represented in different ways by the two media types.

In addition to information that is provided for the benefit of REST clients, some other aspects of a REST design should be specified in a structured way simply for the purpose of fostering consistent design in the implementation. The design and structure of URIs is a particularly important example of this—clients should treat URIs as opaque, but server-side implementations need to understand the URIs they respond to, and URIs often contain metadata that is useful for someone who is learning an API interactively, so consistent structure in URIs is important. The purist side of us would like to leave the structure of URIs completely out of the specification, but in practice, the URI formats often link the specification to the actual implementation. RSDL allow the structure of a URI to be specified, and variable portions of a URI can be specified using a URI template (URI Templates). Our documentation stylesheets omit the URI structure in client API documentation for all resources except the home resource.

This information is highly structured, with many cross references and semantic constraints, but it is typically documented in formats like HTML, Wiki pages, or word processing documents, which offer no aid in getting the structure right. And although this information is often reused in many different ways, it is not generally created in formats that foster reuse. Beyond that, an XML representation of a RESTful web service is extremely useful for teaching RESTful service design, creating and maintain designs over time alongside implementation code, guiding the process of design, visualizing and evaluating service designs created by others, generating various forms of human and machine-readable documentation, providing data for testing clients, and generating stubs for server-side code. The structure of RSDL's schema can guide the design process, nudging designs to be more RESTful[1].

A RESTful service typically needs to document additional information beyond normal HTTP semantics. RSDL allows the following to be documented either in schemas or documentation associated with a mediatype, or in documentation elements, which allow documentation to be written in HTML, DocBook, or other formats, and can be placed throughout a RSDL description. This information includes:

- Application semantics that are orthogonal to REST per se. This is information that is completely unknown to a generic REST client, but this information largely defines what the application is all about. This includes the application semantics of most XML elements or JSON properties. For instance, in Atom, these semantics include the meaning of container elements like feeds, entries, and content, and metadata elements like author, category, and contributor, etc. In the HTML media type, these semantics include all intended browser behavior implied by the data that does not involve the use of links.

- Protocol semantics added by a particular media type. For instance, the Atom Publishing Protocol specifies that a POST to a collection URI adds a new member to a collection, a side-effect that would be difficult for a generic client to discover if it had no knowledge of the media type.

The best-known description language for REST is WADL. WADL provides a detailed, well-structured description of a REST interface, but the structure of WADL is driven by URI patterns and server-side implementation concerns rather than hypermedia-driven design. WADL has been criticized for inviting URI-based tight coupling between the client and the server, so that changes to server-side code can break existing clients. From a REST perspective, the main problem with WADL is that it

[1]It cannot, of course, force a design to be RESTful, and we are confident that RSDL will also be used to create poor designs, but RSDL enforces some important constraints, makes it easier to get some important aspects of RESTful design right, and makes it easier to create RESTful designs collaboratively and verify them using software. A good notation can facilitate good design, but it does not magically produce good design.

exposes interfaces using static metadata, instead of describing conventions for discovering and using links, and even provides tools to create stub code for clients and servers from the WADL description, code that is guaranteed to rely on this static metadata. WADL directly exposes URIs and fixed paths instead of relying on links, documents specific errors instead of relying on generic HTTP processing, and does not distinguish information needed to specify the server from information provided to the client interface.

In many ways, RSDL is similar to WADL, with a great deal of overlap in the information that is represented. But RSDL is designed for hypermedia-driven services. In RSDL, every service has a single published entry point, which corresponds to a home resource, and all other resources can be discovered from the home resource using links. Like WADL, RSDL describes resources and relationships among resources, but RSDL always uses links to describe relationships. Like WADL, RSDL describes the methods that can be used for to a given resource, but RSDL does not describe expected error codes for each method; instead, it relies on the use of generic HTTP status codes; if specific semantics are required, e.g. to document the use of a redirect for authentication purposes, RSDL allows the semantics of a given status code to be documented globally for the service. Like WADL, RSDL allows schemas that describe expected formats for representations, but RSDL also allows descriptions in other formats, including W3C XML Schemas, Relax-NG, JSON Schema, HTML descriptions, text-based descriptions, or anything else that can be referred to with a URI.

A Sample RSDL Description

The following RSDL example describes a simple service that supports a collection of documents. The service uses several media types: an XML Home document as the entry point, an atom feed to represent the collection of documents, an HTML description that describes the service, and a media type for the documents used in the service. The schema for RSDL is provided in an appendix. Documentation is embedded at many points in the description, using a separate documentation module that is included. The appendixes provide documentation modules for HTML and DocBook, this example uses the HTML module.

The Documents Service

```
<?xml version="1.0" encoding="utf-8"?>
<service name="Documents"
  identity-provider-ref="idp"
  xmlns="http://identifiers.emc.com/rsdl"
  xmlns:html="http://www.w3.org/1999/xhtml/">

  <documentation>This RESTful service provides a simple ATOM feed that allows do
    created, modified, or deleted. </documentation>

  <start ref="res-home"/>

  <media-types>
    <media-type id="med-document" name="application/vnd.example.document+xml">
      <documentation> The media type for the service described by this RSDL desc
      <description href="example.com/mediatypes/documents.rnc" type="rnc"/>
    </media-type>
    <media-type id="med-home-xml" name="application/home+xml">
      <documentation> Home Documents for HTTP Services: XML Syntax </documentati
      <description href="http://tools.ietf.org/html/draft-wilde-home-xml-01.htm
    </media-type>
    <media-type id="med-atom" name="application/atom+xml">
      <documentation> Atom feeds, updateable using AtomPub conventions, with fe
      </documentation>
    </media-type>
    <media-type id="med-html" name="text/html">
```

```
    <documentation> HTML documents. </documentation>
  </media-type>
</media-types>

<resources>
  <resource id="res-home" name="home">
    <location uri="/"/>
    <links>
      <link link-relation-ref="rel-documents" resource-ref="res-documents"/>
      <link link-relation-ref="rel-about" resource-ref="res-about"/>
    </links>
    <methods>
      <method name="GET">
        <response>
          <representation media-type-ref="med-home-xml" entity="resources"/>
        </response>
      </method>
    </methods>
  </resource>

  <resource id="res-documents" name="documents">
    <location uri="/documents"/>
    <links>
      <link link-relation-ref="rel-self" resource-ref="res-documents"/>
      <link link-relation-ref="rel-first" resource-ref="res-documents"/>
      <link link-relation-ref="rel-next" resource-ref="res-documents"/>
      <link link-relation-ref="rel-last" resource-ref="res-documents"/>
      <link link-relation-ref="rel-previous" resource-ref="res-documents"/>

      <link link-relation-ref="rel-alternate" resource-ref="res-document">
        <documentation>
          In an ATOM feed, we use the alternate link relation to indicate the
          type.
        </documentation>
      </link>
    </links>
    <methods>
      <method name="GET">
        <response>
          <representation media-type-ref="med-atom" entity="atom:feed"/>
        </response>
      </method>
      <method name="POST">
        <request>
          <documentation>Creates a document</documentation>
          <representation media-type-ref="med-document" entity="document"/>
        </request>
        <response>
          <documentation>Returns the newly created document</documentation>
          <representation media-type-ref="med-document" entity="document"/>
        </response>
      </method>
    </methods>
  </resource>

  <resource id="res-document" name="document">
    <documentation> </documentation>
    <location template="/document/{oid}">
```

```
        <var name="oid">
          <documentation> Identifier for the document. </documentation>
        </var>
      </location>
      <links>
        <link link-relation-ref="rel-self" resource-ref="res-document"/>
      </links>
      <methods>
        <method name="GET">
          <response>
            <representation media-type-ref="med-document" entity="res-document"
          </response>
        </method>

        <method name="PUT">
          <request>
            <representation media-type-ref="med-document" entity="res-document"
          </request>
        </method>

        <method name="DELETE"/>

      </methods>
    </resource>

    <resource id="res-about" name="about" public="true">
      <documentation>An HTML page describing the service.</documentation>
      <location uri="/about"/>
      <methods>
        <method name="GET">
          <response>
            <representation media-type-ref="med-html" entity="html"/>
          </response>
        </method>
      </methods>
    </resource>

  </resources>

  <link-relations>
    <documentation> Link relations used in the media-type. IANA link relations
        uri="http://www.iana.org/assignments/link-relations/link-relations.xml"
      link relations will be registered on identifiers.example.com. </documenta

    <link-relation id="rel-about" name="about"/>
    <link-relation id="rel-alternate" name="alternate"/>
    <link-relation id="rel-self" name="self"/>
    <link-relation id="rel-edit" name="edit"/>
    <link-relation id="rel-first" name="first"/>
    <link-relation id="rel-last" name="last"/>
    <link-relation id="rel-next" name="next"/>
    <link-relation id="rel-previous" name="previous"/>
    <link-relation id="rel-parent" name="parent"/>

    <link-relation id="rel-documents" name="identifiers.example.com/linkrel/doc

  </link-relations>
```

```
  <headers>
    <header id="hea-authenticate" name="WWW-Authenticate" type="request"/>
  </headers>

  <authentication>
    <mechanism id="aut-http" name="HTTP Authentication" authentication-type="rf
      <documentation>
        We use HTTP Authentication as defined in <ref uri="http://tools.ietf.or
        with custom schemes. If an unauthenticated user tries to access a prote
        <ref status-code="sta-unauthorized"/> status is returned, along with on
        <ref header="hea-authenticate"/> headers, each of which contains an aut
        <html:em>challenge</html:em>. The challenges consist of a scheme follow
      </documentation>
      <scheme name="basic">
        <parameter name="realm"/>
      </scheme>
    </mechanism>
    <identity-provider id="idp" mechanism-ref="aut-http"/>
  </authentication>

  <status-codes>
    <status id="sta-unauthorized" code="401">
      <documentation> The request requires authentication. When this status cod
        response contains one or more <ref header="hea-authenticate"/> headers
          <ref mechanism="aut-http">authentication challenge</ref>. </documenta
    </status>
  </status-codes>
</service>
```

This example will be examined in more detail in the following section.

The Structure of a RSDL Description

In this section, we will explore the structure of a RSDL description, using the example from the previous section.

Service

The top-level element for a RSDL description is the service element. Here is the structure of the service element, in Relax-NG Compact syntax, together with the optional attribute used to designate an identity provider for authentication, and a start element that identifies the home resource.

```
start = service
service =
    element service {
        id?,
        name,
        identity-provider-ref?,
        documentation?,
        service-start,
        media-types,
        resources,
        link-relations?,
        headers?,
        authentication?,
```

```
        status-codes?,
        uri-parameters?
    }

service-start = element start { idref }
idref = attribute ref { xsd:IDREF }

identity-provider-ref = attribute identity-provider-ref { xsd:IDREF }
```

Consider the following fragment:

```
<service name="Documents"
  identity-provider-ref="idp"
  xmlns="http://identifiers.emc.com/rsdl"
  xmlns:html="http://www.w3.org/1999/xhtml/">

  <documentation> This RESTful service provides a simple ATOM feed that allows
    created, modified, or deleted. </documentation>

  <start ref="res-home"/>

  !!! SNIP !!!

 </service>
```

The `name` of a service is provided primarily for human beings who need to refer to the service. REST clients do not use a name to identify a service, they enter a service using the URI of the home resource.[2]

The `identity-provider-ref` attribute refers to an identity provider that can be used to authenticate for the service. This attribute is an `IDREF` that refers to an identity provider in the `authentication` element, which is discussed later. If an identity provider is specified, all resources require authentication by default, unless declared public (this is described in the section on `resources`). If no identity provider is specified, resources do not require authentication by default, but an individual resource can specify an identity provider.

The `start` element identifies the home resource, which is the published entry point for the service. Because every RESTful service should have a single published entry point, the `start` element is required.

Media Types

In RSDL, media types describe the resource representations used in a REST service. In the systems we design, it is quite common for a service to use multiple media types; for instance, the example we are discussing is focused on documents, but it also uses XML Home Documents to provide a set of available links in the home resource, uses Atom and AtomPub with feeds to represent collections, and uses HTML to provide a human-readable description of the service. The resource representations for a media type may be in XML, for which there are standard schema languages, but they may also use JSON or any other format, and the authoritative description for a given media type may be a document available in HTML, text, PDF, or some other format. Some media types are shared among multiple specifications, other media types have no authoritative description. Two media types may represent the same properties in XML and JSON respectively. Because of this, media type descriptions in RSDL allow reference to any available schema or description in formats we use, but they do not require a schema or a description, and a media type can be documented directly in its `documentation`

[2]Some REST designers object to naming services or resources because the names are not part of the interface. We believe that names are important because human beings need a convenient way to refer to things that they are discussing, and they can be helpful metadata in some systems that use RSDL descriptions.

element. Because some services need both XML and JSON representations, we also allow RSDL descriptions to avoid duplication by specifying properties in resources rather than in the media types, and documenting the manner in which properties are mapped to the corresponding `+xml` or `+json` media types. This is described in the section on `resources`.

The following schema fragment describes RSDL media types.

```
media-types = element media-types { documentation?, media-type* }
media-type = element media-type { id?, name, documentation?, description* }
media-type-ref = attribute media-type-ref { xsd:IDREF }

description = element description { type, href, documentation? }
type = attribute type { "rnc" | "rng" | "xsd" | "JSONSchema" | "sedola" | "text
href = attribute href { xsd:anyURI }
```

The `type` attribute of a media type description refers to the language of the document referred to by the `href` attribute. It can be an XML schema (in Relax-NG Compact Syntax (`rnc`), Relax-NG (`rng`), or W3C XML Schema (`xsd`), a JSON Schema (`JSONSchema`), a human readable description (in `text` or `html`), or Sedola (`sedola`), an XML format that provides a structured description of a media type with cross-references into the document that defines it.[3]

Here are the media types for our example:

```
<media-types>
  <media-type id="med-document" name="application/vnd.example.document+xml">
    <documentation> The media type for the service described by this RSDL des
    <description href="example.com/mediatypes/documents.rnc" type="rnc"/>
  </media-type>
  <media-type id="med-home-xml" name="application/home+xml">
    <documentation> Home Documents for HTTP Services: XML Syntax </documentat
    <description href="http://tools.ietf.org/html/draft-wilde-home-xml-01.htm
  </media-type>
  <media-type id="med-atom" name="application/atom+xml">
    <documentation> Atom feeds, updateable using AtomPub conventions, with fe
    </documentation>
  </media-type>
  <media-type id="med-html" name="text/html">
    <documentation> HTML documents. </documentation>
  </media-type>
</media-types>
```

Note that the `media-type` for `application/vnd.example.document+xml` provides both inline documentation and a schema, the `media-type` for `application/home+xml` provides inline documentation and a reference to an HTML page, and the `media-type`s for `application/atom+xml` and `text/html` provide only brief inline documentation.

Resources

Resources are at the heart of RSDL, which provides rich support for them. Here is a schema fragment for resources:

```
resources = element resources { id?, documentation?, resource* }
```

[3]We use Sedola to describe standard media types such as Atom and HTML in the same structured way. If we were writing these media types ourselves, we would use RSDL to specify the service, and generate Sedola from the RSDL description.

```
resource =
    element resource {
        documentation?,
        id,
        name,
        identity-provider-ref?,
        public?,
        status?,
        extends?,
        location?,
        properties?,
        links?,
        methods
    }

name = attribute name { text }
public = attribute public { "true" }

status = implementation-status?, design-status?
implementation-status =
    attribute implementation-status {
        "future" | "assigned" | "poc" | "partial" | "complete" | "passed"
    }
design-status =
    attribute design-status { "future" | "assigned" | "poc" | "partial" | "comp

# A resource can extend an existing resource definition, inheriting what it alr
extends = attribute extends { xsd:IDREF }

location = element location { documentation?, (uri | (uri-template, var*)) }
uri = attribute uri { xsd:anyURI }
uri-template = attribute template { xsd:string }
# uri-parameter-ref indicates that the value is supplied by the client, using a
# If no uri-parameter-ref is present, the value is supplied by the server.
var = element var { documentation?, id?, name, uri?, uri-parameter-ref? }

properties = element properties { documentation?, property* }
property = element property { id?, name, documentation? }

links = element links { documentation?, link* }
link = element link { link-relation-ref, resource-ref, status?, documentation? }
resource-ref = attribute resource-ref { xsd:IDREF }
link-relation-ref = attribute link-relation-ref { xsd:anyURI }

methods = element methods { method* }
method = element method { id?, method-name, status?, request?, response? }
method-name = attribute name { http-method }

request = element request { documentation?, request-uri-parameters?, header-ref
response =
    element response { documentation?, response-status-codes?, header-refs?, re
request-uri-parameters = element uri-parameters { request-uri-parameter }
request-uri-parameter = element uri-parameter { idref }
# When possible, avoid documenting status codes at the individual request level
# If you have to, declare it globally and refer to it from the request.
response-status-codes =
    element status-codes {
        element status-code { ref }*
```

```
    }
# When possible, avoid documenting headers at the individual request level.
# If you have to, declare it globally and refer to it from the request.
header-refs = element header-refs { documentation?, header-ref* }
header-ref = element header-ref { ref }
representation = element representation { documentation?, media-type-ref, entity
entity = attribute entity { text }
http-method = "GET" | "PUT" | "HEAD" | "POST" | "DELETE" | "TRACE" | "OPTIONS"
```

Let's look at three resources from our example. The home resource for our example is an XML Home Document. In our own designs, we frequently use JSON Home Documents, or XML Home Documents as the home document for a resource, but Atom feeds are also popular as home documents, and some services use a single application-domain object as the home document. Here is the RSDL description of an XML Home Document resource:

```
<resource id="res-home" name="home">
  <location uri="/"/>
  <links>
    <link link-relation-ref="rel-self" resource-ref="res-documents"/>
    <link link-relation-ref="rel-documents" resource-ref="res-documents"/>
    <link link-relation-ref="rel-about" resource-ref="res-about"/>
  </links>
  <methods>
    <method name="GET">
      <response>
        <representation media-type-ref="med-home-xml" entity="resources"/>
      </response>
    </method>
  </methods>
</resource>
```

In this example, the methods and the links define the interface for the resource. Like the name of a service, the name of a resource is not part of the interface. Neither is the `id`, it exists to allow cross-reference in a RSDL description. Nor is the `location` element, which is optional.

A method specifies an HTTP method name that is supported for the resource in which it contains and the `request` or `response` associated with the method. In the above example, a `GET` method results in a response that contains the representation of an XML Home Document, which was specified in the media types we discussed in an earlier section.

The links each define a transition to another resource. Each link contains a reference to a globally declared link relation, discussed later, and the resource to which the link resolves. One link element is created for each (`link-relation-ref`, `resource-ref`) pair. Note that RSDL allows links to be defined only in terms of link relations, not in terms of the structure of URIs. A link refers to a resource, not a representation; to find the available representations for a resource, look at the resource and see what it returns for `GET`. In this example, the second link contains a reference to a globally declared link relation with the id `rel-documents`, which is defined as follows:

```
  <link-relation id="rel-documents" name="identifiers.example.com/linkrel/d
```

It also contains a reference to the resource with the id `res-documents`, an Atom feed which is described below:

```
<resource id="res-documents" name="documents">
  <location uri="/documents"/>
```

```
      <links>
        <link link-relation-ref="rel-self" resource-ref="res-documents"/>

        <link link-relation-ref="rel-first" resource-ref="res-documents"/>
        <link link-relation-ref="rel-next" resource-ref="res-documents"/>
        <link link-relation-ref="rel-last" resource-ref="res-documents"/>
        <link link-relation-ref="rel-previous" resource-ref="res-documents"/>

        <link link-relation-ref="rel-alternate" resource-ref="res-document">
          <documentation>
            In an ATOM feed, we use the alternate link relation to indicate the
            type.
          </documentation>
        </link>
      </links>
      <methods>
        <method name="GET">
          <response>
            <representation media-type-ref="med-atom" entity="atom:feed"/>
          </response>
        </method>
        <method name="POST">
          <request>
            <documentation>Creates a document</documentation>
            <representation media-type-ref="med-document" entity="document"/>
          </request>
          <response>
            <documentation>Returns the newly created document</documentation>
            <representation media-type-ref="med-document" entity="document"/>
          </response>
        </method>
      </methods>
    </resource>
```

This resource is an Atom feed that supports paging and adding new documents. The following link relations refer to the IANA standard link relations used for paging:

```
<link-relation id="rel-first" name="first"/>
<link-relation id="rel-last" name="last"/>
<link-relation id="rel-next" name="next"/>
<link-relation id="rel-previous" name="previous"/>
```

Because this resource supports AtomPub, it has a POST method. Both the request and the response refer to the `document` entity, as defined by the media type with the identifier `med-document`.

```
<method name="POST">
    <request>
        <documentation>Creates a document</documentation>
        <representation media-type-ref="med-document" entity="document"/>
    </request>
    <response>
        <documentation>Returns the newly created document</documentation>
        <representation media-type-ref="med-document" entity="document"/>
    </response>
</method>
```

This resource uses the IANA standard `alternate` link relation to describe the type of the collection's members:

```
<link link-relation-ref="rel-alternate" resource-ref="res-document">
   <documentation>
       In an ATOM feed, we use the alternate link relation to indicate the membe
   </documentation>
</link>
```

Here is the resource that corresponds to the member type:

```
    <resource id="res-document" name="document">
      <location template="/document/{oid}">
        <var name="oid">
          <documentation> Identifier for the document. </documentation>
        </var>
      </location>
      <links>
        <link link-relation-ref="rel-self" resource-ref="res-document"/>
      </links>
      <methods>
        <method name="GET">
          <response>
            <representation media-type-ref="med-document" entity="res-document"
          </response>
        </method>

        <method name="PUT">
          <request>
            <representation media-type-ref="med-document" entity="res-document"
          </request>
        </method>

        <method name="DELETE"/>

      </methods>
    </resource>
```

Sometimes a resource has different authentication constraints than other resources in the same service. In our example, the `about` resource does not require authentication, even though the service has specified authentication globally. This is expressed using the `public=true` attribute:

```
<resource id="res-about" name="about" public="true">
    <documentation>An HTML page describing the service.</documentation>
    <location uri="/about"/>
    <methods>
        <method name="GET">
            <response>
                <representation media-type-ref="med-html" entity="html"/>
            </response>
        </method>
    </methods>
```

```
</resource>
```

The opposite situation is also common - a service may need no authentication on most resources, but require authentication for specific resources such as a payment resource, or perhaps one resource uses a different kind of authentication from others. Both can be expressed using an `identity-provider-ref` attribute on the resource, specifying the authentication to be used.

The `location` element is always optional, and is not part of the client interface. It specifies a location URI or a URI template that describes the format of a URI that resolves to a given resource. Here are two location elements that illustrate both possibilities:

```
<location uri="/about"/>
```

```
<location template="/document/{oid}">
    <var name="oid">
        <documentation> Identifier for the document. </documentation>
    </var>
</location>
```

A variable in a URI template sometimes corresponds to a URI parameter that should be provided by the client. This can be expressed by adding a `uri-parameter-ref` attribute to the variable:

```
<location template="/document/{oid}">
    <var name="oid" uri-parameter-ref="par-oid"/>
</location>
```

URI parameters can also be specified in a request. For instance, if we want to allow an Atom feed to be sorted, we could provide a URI parameter to specify the sort in the `GET` request:

```
<method name="GET">
    <request>
        <uri-parameters>
            <uri-parameter ref="par-sortby"/>
        </uri-parameters>
    </request>
    <response>
        <representation media-type-ref="med-atom" entity="feed"/>
    </response>
</method>
```

On a `resource` element, the `extends` attribute allows one resource to inherit all items from an existing resource and extend it by adding further items or override existing items. For instance, the following description creates an `invoice` resource that inherits everything found in a `document` resource, and adds a link to represent the customer for an invoice:

```
<resource id="res-invoice" name="invoice" extends="res-document">
  <links>
    <link link-relation-ref="rel-customer" resource-ref="res-customer"/>
  </links>
</resource>
```

Status attributes on a `resource` element can be used to track both the design status and the implementation status:

```
<resource id="res-document" name="document" design-status="approved" implementa
```

`property` elements can be used to specify properties of a resource. This is typically done when more than one media type represents the same resource, in order to minimize redundant specification. In this case, the media types should describe how a property is mapped to its representation. Note that property elements are defined purely by documentation, they do not have a standard type system or a standard set of datatypes.

```
<properties>
    <property name="expiration">
        <documentation>A date, representing the expiration date of the document
    </property>
</properties>
```

Link Relations

Link relations are declared globally. We recommend that even IANA standard link relations be documented for a service, because there are many standard link relations, and their meaning is not completely specified, their meaning needs to be specified concretely for a given service. Link relations are declared globally and referred to from individual resources. If the name of a link relation is not registered with IANA, its name should be a URI, such as `identifiers.example.com/linkrel/documents`.

```
  <link-relations>
    <documentation> Link relations used in the media-type. IANA link relations
        uri="http://www.iana.org/assignments/link-relations/link-relations.xml"
      link relations will be registered on identifiers.example.com. </documenta

    <link-relation id="rel-about" name="about"/>
    <link-relation id="rel-alternate" name="alternate"/>
    <link-relation id="rel-self" name="self"/>
    <link-relation id="rel-edit" name="edit"/>
    <link-relation id="rel-first" name="first"/>
    <link-relation id="rel-last" name="last"/>
    <link-relation id="rel-next" name="next"/>
    <link-relation id="rel-previous" name="previous"/>
    <link-relation id="rel-parent" name="parent"/>

    <link-relation id="rel-documents" name="identifiers.example.com/linkrel/doc

  </link-relations>
```

Headers

Headers are declared globally. We recommend that even standard HTTP headers be documented for a service, so that clients know what is required.

```
  <headers>
    <header id="hea-authenticate" name="WWW-Authenticate" type="request"/>
```

```
  </headers>
```

Authentication

Authentication mechanisms and identity providers are declared globally. Here is the schema fragment for authentication:

```
authentication = element authentication { mechanism*,  identity-provider* }
mechanism = element mechanism { id?, name, authentication-type, documentation?,
mechanism-ref = attribute mechanism-ref { xsd:IDREF }
identity-provider = element identity-provider { id, mechanism-ref }
authentication-type = attribute authentication-type { text }
scheme = element scheme { id?, name, documentation?, scheme-parameter* }
scheme-parameter = element parameter { id?, name, documentation? }
```

Here is the authentication used in our example:

```
  <authentication>
    <mechanism id="aut-http" name="HTTP Authentication" authentication-type="rf
      <documentation>
        We use HTTP Authentication as defined in <ref uri="http://tools.ietf.or
        with custom schemes. If an unauthenticated user tries to access a prote
        <ref status-code="sta-unauthorized"/> status is returned, along with on
        <ref header="hea-authenticate"/> headers, each of which contains an aut
        <html:em>challenge</html:em>. The challenges consist of a scheme follow
      </documentation>
      <scheme name="basic">
        <parameter name="realm"/>
      </scheme>
      <identity-provider id="idp" mechanism-ref="aut-http"/>
    </mechanism>
  </authentication>
```

Status Codes

Status codes are declared globally. In order to avoid tight coupling, we do not provide a direct mechanism for documenting status codes for an individual request (the documentation for a request can refer to globally declared status codes as necessary). We recommend that a service document any uncommon status codes or uncommon uses of status codes at the service level so that clients can be prepared to handle them. Details of error conditions can be reported with Problem Details for HTTP APIs conventions using the `http-problem` element, as specified in the schema

```
status-codes = element status-codes { documentation?, status-code* }
status-code = element status { code, id, documentation?, http-problem? }
status-code-ref = attribute ref { xsd:IDREF }
code = attribute code { HTTP-status-enum }
HTTP-status-enum =
    "100"
    | "101"
    | "102"
    | "200"
    | "201"
    | "203"
```

```
        | "204"
        | "205"
        | "206"
        | "207"
        | "208"
        | "301"
        | "302"
        | "303"
        | "304"
        | "305"
        | "306"
        | "307"
        | "308"
        | "400"
        | "401"
        | "402"
        | "403"
        | "404"
        | "405"
        | "406"
        | "407"
        | "408"
        | "409"
        | "410"
        | "411"
        | "412"
        | "413"
        | "414"
        | "415"
        | "416"
        | "417"
        | "418"
        | "420"
        | "422"
        | "423"
        | "424"
        | "425"
        | "426"
        | "428"
        | "429"
        | "431"
        | "444"
        | "449"
        | "450"
        | "451"
        | "494"
        | "495"
        | "496"
        | "497"
        | "499"
        | "500"
        | "501"
        | "502"
        | "503"
        | "504"
        | "505"
        | "506"
        | "507"
```

```
  | "508"
  | "509"
  | "510"
  | "511"
  | "598"
  | "599"
```

URI Parameters

URI parameters are declared globally, and can be referenced from `request`s in `method`s.

```
<uri-parameters>
  <uri-parameter id="par-sortby" name="sortby" datatype="string">
    <documentation>
      Specifies a sort order for the collection.
    </documentation>
  </uri-parameter>
</uri-parameters>
```

Documentation Modules

The content allowed within a `documentation` element is defined by a separate module that is imported.

```
include "documentation.rnc"
```

We have defined modules for HTML and for DocBook. A `documentation` element can have a title, which is treated much like a section header, and an `inline` attribute. Documentation that contains code examples, tables, multiple sections, or large amounts of text can be marked `inline="false"` so that a stylesheet can convert it to endnotes, a popup window, or whatever representation is appropriate to the format the stylesheet produces.

Here is the first part of the HTML documentation module:

```
namespace html = "http://www.w3.org/1999/xhtml/"
namespace docbook = "http://docbook.org/ns/docbook"

documentation = element documentation { inline?, doc-title?, html }

inline = attribute inline { ( "true" | "false" ) }
doc-title = element title { text }

html =  html-content*
html-content = (html-element | text | ref)
html-element = element html:* { html-attribute*, html-content* }
html-attribute = attribute * { text? }
```

Documentation frequently needs to refer to items from the RSDL description itself, so we support typed references interspersed with documentation:

```
ref = element ref {
  (  attribute idref { xsd:IDREF }
  | attribute uri { xsd:anyURI }
```

```
  | attribute media-type { xsd:IDREF }
  | attribute header { xsd:IDREF }
  | attribute mechanism { xsd:IDREF }
  | attribute identity-provider { xsd:IDREF }
  | attribute scheme { xsd:IDREF }
  | attribute scheme-parameter { xsd:IDREF }
  | attribute status-code { xsd:IDREF }
  | attribute uri-parameter { xsd:IDREF }
  | attribute resources { xsd:IDREF }
  | attribute resource { xsd:IDREF }
  | attribute var { xsd:IDREF }
  | attribute property { xsd:IDREF }
  | attribute header { xsd:IDREF }
  | attribute method { xsd:IDREF }
  ),
  text?  # Uses the name of the referred item if not provided
}
```

We have already seen an example that uses `ref` elements in the `documentation` element for a `mechanism`:

```
  <authentication>
    <mechanism id="aut-http" name="HTTP Authentication" authentication-type="rf
      <documentation>
        We use HTTP Authentication as defined in <ref uri="http://tools.ietf.or
        with custom schemes. If an unauthenticated user tries to access a prote
        <ref status-code="sta-unauthorized"/> status is returned, along with on
        <ref header="hea-authenticate"/> headers, each of which contains an aut
        <html:em>challenge</html:em>. The challenges consist of a scheme follow
      </documentation>
      <scheme name="basic">
        <parameter name="realm"/>
      </scheme>
      <identity-provider id="idp" mechanism-ref="aut-http"/>
    </mechanism>
  </authentication>
```

Initial Experience

RSDL is quite new. We have recently started using it for some REST projects in the Information Intelligence Group at EMC Corporation. In this section, we would like to share our experiences so far.

We have found RSDL a valuable teaching tool in the hands of a teacher who is comfortable editing XML. On two projects, one of the authors was involved with an early design that was having difficulty ramping up the design, and responded by editing a RSDL description on a shared screen during a phone call with an engineer from the project. In each case, it was easy to copy similar designs from existing RSDL descriptions and modify them, then provide the RSDL description to the engineer as an initial skeleton for the design. With a standard description language, we believe we will be able to grow a library of common REST design patterns that can be copied and modified to design a REST interface.

Before RSDL, we maintained our REST specifications largely in Wikis, and a single design was frequently distributed across many Wiki pages. As a specification matured, it was difficult to keep these pages in sync, with accurate cross-references. More than one page might purport to represent the same design, different pages might represent the same information in more than one way, and the Wiki pages did little to enforce REST conventions. RSDL makes it much easier to version a design. By checking a RSDL description into source code control, reviewers can see changes in the design

specification along with code changes, and it is always possible to retrieve the design specification that corresponds to a particular version of the code.

Before RSDL, user documentation was created from information in these Wikis, initially by converting it to DocBook. RSDL allows this to be done using an XSLT transformation. We expect to integrate RSDL into our documentation tool chain.

While the structure of a URI should not be provided to clients, we have found that it is important for actually implementing a service. For instance, if the URI structure is included, skeleton code could be generated to create stubs that can be used to implement a server, using a REST framework such as JAX-RS for Java.

We have been experimenting with REST clients that perform QA testing using a RSDL description. One of us wrote a REST client that takes a RSDL file as input and issues HTTP requests, navigating the service, and ensuring that each response corresponds to the RSDL description and reporting any deviations. In effect, the test client infers the design from the actual implementation, and compares it to the design specified by the RSDL description. As a result, we found missing resources and links in the implementation, and resources that had been implemented but not specified, and methods that should not be available on a resource. We also found code that did not properly handle edge cases for data provided via URI templates (in the future, we may develop a fuzzer to test for this kind of problem). We have used this test client in our build process. When a new resource is implemented, or an existing one is changed, we can test to see if the RSDL was also updated appropriately. If not, the build fails. Because both the build process and the documentation are synchronized with the specification, we can guarantee that the specification, the code, and the documentation are kept in sync.

Future Work

This section describes some tools that have not yet been written, or in very early stages, that we are considering implementing.

RSDL Lint is an XSLT transform that checks a RSDL description for internal consistency. In its current form, it ensures that an `IDREF` refers to an element of the appropriate type, ensures that identifiers follow our in-house standards, screens for media types that we do not allow in our interfaces, and warns of items that cannot be reached from the home resource by navigation. It can be integrated into the build process, raising either errors or warnings. We expect to expand the functionality of RSDL Lint significantly.

When documenting REST APIs, we often want to capture the results of concrete scenarios running on a server and show the entire sequence of calls, including the URIs, requests, and responses. We are writing a tool executes a message flow using an XML vocabulary that refers to the components of a RSDL description. This tool generates HTML to document the requests and responses of the message flow. As the specification evolves, or the data in the server changes, the message flow can be run again to create up-to-date documentation.

We plan to experiment with domain-specific editors and browsers for RSDL, including the ability to select templates that implement standard RESTful design patterns so that they can be modified.

Conclusions

RSDL provides a formal notation that can be used to think more clearly about the model for a RESTful service. It is an XML vocabulary designed for loosely-coupled, hypermedia-driven RESTful design, and supports the rich structure and cross references of a REST service. We find it useful as a teaching tool, a design tool, and a way to represent known design patterns that can be copied and modified in a new RESTful service. RSDL can be used to generate documentation in various formats.

As a machine-readable description, RSDL can be used for software-based testing of design consistency constraints, dynamic testing of the implementation based on the design, and in a variety of other tools. These tools can be integrated with the build process, identifying mismatches between the specification and the implementation. By versioning a RSDL description together with source code, reviewers can

see changes in the design specification along with code changes, and it is always possible to retrieve the design specification that corresponds to a particular version of the code.

A. RSDL Schema

The following schema, in RELAX-NG compact syntax, defines RSDL.

```
default namespace rsdl = "http://identifiers.emc.com/rsdl"

include "documentation.rnc"

start = service
service =
    element service {
        id?,
        name,
        identity-provider-ref?,
        documentation?,
        service-start,
        media-types,
        resources,
        link-relations?,
        headers?,
        authentication?,
        status-codes?,
        uri-parameters?
    }

id = attribute id { xsd:ID }
idref = attribute ref { xsd:IDREF }

service-start = element start { idref }
identity-provider-ref = attribute identity-provider-ref { xsd:IDREF }

media-types = element media-types { documentation?, media-type* }
media-type = element media-type { id?, name, documentation?, description* }
media-type-ref = attribute media-type-ref { xsd:IDREF }
description = element description { type, href, documentation? }
type = attribute type { "rnc" | "rng" | "xsd" | "JSONSchema" | "sedola" | "text
href = attribute href { xsd:anyURI }

resources = element resources { id?, documentation?, resource* }
resource =
    element resource {
        documentation?,
        id,
        name,
        identity-provider-ref?,
        public?,
        status?,
        extends?,
        location?,
        properties?,
        links?,
```

```
        methods
    }

name = attribute name { text }
public = attribute public { "true" }

status = implementation-status?, design-status?
implementation-status =
    attribute implementation-status {
        "future" | "assigned" | "poc" | "partial" | "complete" | "passed"
    }
design-status =
    attribute design-status { "future" | "assigned" | "poc" | "partial" | "comp
# A resource can extend an existing resource definition, inheriting what it alr

extends = attribute extends { xsd:IDREF }

location =
    element location {
        documentation?,
        (uri | (uri-template, var*))
    }
uri-template = attribute template { xsd:string }
# URI refers to a link relation. If absent, it is a local link relation, identi
# uri-parameter-ref indicates that the value is supplied by the client, using a
#    If no uri-parameter-ref is present, the value is supplied by the server.
var = element var { documentation?, id?, name, uri?, uri-parameter-ref? }

properties = element properties { documentation?, property* }
property = element property { id?, name, documentation? }

links = element links { documentation?, link* }
link = element link { link-relation-ref, resource-ref, status?, documentation? }
resource-ref = attribute resource-ref { xsd:IDREF }
link-relation-ref = attribute link-relation-ref { xsd:anyURI }

methods = element methods { method* }
method = element method { id?, method-name, status?, request?, response? }
method-name = attribute name { http-method }
request = element request { documentation?, request-uri-parameters?, header-refs
response =
    element response { documentation?, response-status-codes?, header-refs?, rep
request-uri-parameters = element uri-parameters { request-uri-parameter }
request-uri-parameter = element uri-parameter { idref }
response-status-codes =
    element status-codes {
        element status-code { ref }*
    }

header-refs = element header-refs { documentation?, header-ref* }
header-ref = element header-ref { ref }

representation = element representation { documentation?, media-type-ref, entity
entity = attribute entity { text }

http-method = "GET" | "PUT" | "HEAD" | "POST" | "DELETE" | "TRACE" | "OPTIONS"

link-relations = element link-relations { documentation?, link-relation* }
```

```
link-relation = element link-relation { documentation?, id, status, link-relati
link-relation-name = attribute name { xsd:anyURI }

headers = element headers { header* }
header = element header { id?, name, header-type, documentation? }
header-type = attribute type { "request" | "response" | "general" | "entity" }

authentication = element authentication { mechanism*, identity-provider? }
mechanism = element mechanism { id?, name, authentication-type, documentation?,
mechanism-ref = attribute mechanism-ref { xsd:IDREF }
identity-provider = element identity-provider { id, mechanism-ref }
authentication-type = attribute authentication-type { text }
scheme = element scheme { id?, name, documentation?, scheme-parameter* }
scheme-parameter = element parameter { id?, name, documentation? }

status-codes = element status-codes { documentation?, status-code* }
status-code = element status { code, id, documentation?, http-problem? }
status-code-ref = attribute ref { xsd:IDREF }
code = attribute code { HTTP-status-enum }
HTTP-status-enum =
    "100"
    | "101"
    | "102"
    | "200"
    | "201"
    | "203"
    | "204"
    | "205"
    | "206"
    | "207"
    | "208"
    | "301"
    | "302"
    | "303"
    | "304"
    | "305"
    | "306"
    | "307"
    | "308"
    | "400"
    | "401"
    | "402"
    | "403"
    | "404"
    | "405"
    | "406"
    | "407"
    | "408"
    | "409"
    | "410"
    | "411"
    | "412"
    | "413"
    | "414"
    | "415"
    | "416"
    | "417"
    | "418"
```

```
        | "420"
        | "422"
        | "423"
        | "424"
        | "425"
        | "426"
        | "428"
        | "429"
        | "431"
        | "444"
        | "449"
        | "450"
        | "451"
        | "494"
        | "495"
        | "496"
        | "497"
        | "499"
        | "500"
        | "501"
        | "502"
        | "503"
        | "504"
        | "505"
        | "506"
        | "507"
        | "508"
        | "509"
        | "510"
        | "511"
        | "598"
        | "599"

    http-problem = element problem { problemType, title, detail, supportId, more }
    problemType = element problemType { xsd:anyURI }
    title = element title { text }
    detail = element detail { text }
    supportId = element supportId { xsd:anyURI }
    more = element more { foreign-element* }
    foreign-element = element * - rsdl:* { any-attribute*, (foreign-element* | text
    any-attribute = attribute * { text? }

    uri-parameters = element uri-parameters { documentation?, uri-parameter* }
    uri-parameter =
        element uri-parameter { id?, name, documentation, datatype, value-range?, d
    uri-parameter-ref = attribute uri-parameter-ref { xsd:IDREF }

    datatype =
        attribute datatype {
            "string"
            | "boolean"
            | "decimal"
            | "float"
            | "double"
            | "duration"
            | "dateTime"
            | "time"
            | "date"
```

```
        | "hexBinary"
        | "base64Binary"
        | "anyURI"
        | "integer"
        | "language"
        | "ID"
        | "IDREF"
        | "integer"
        | "long"
        | "short"
        | "byte"
    }

value-range = element value-range { text }
default-value = element default { text }

uri = attribute uri { xsd:anyURI }

# ref elements are used in the documentation modules
ref =
    element ref {
        ((attribute idref { xsd:IDREF }
          | attribute uri { xsd:anyURI }
          | attribute media-type { xsd:IDREF }
          | attribute header { xsd:IDREF }
          | attribute mechanism { xsd:IDREF }
          | attribute identity-provider { xsd:IDREF }
          | attribute scheme { xsd:IDREF }
          | attribute scheme-parameter { xsd:IDREF }
          | attribute status-code { xsd:IDREF }
          | attribute uri-parameter { xsd:IDREF }
          | attribute resources { xsd:IDREF }
          | attribute resource { xsd:IDREF }
          | attribute var { xsd:IDREF }
          | attribute property { xsd:IDREF }
          | attribute header { xsd:IDREF }
          | attribute method { xsd:IDREF }),
         text?)
        # Uses the name of the referred item if not provided

    }
```

B. HTML module for RSDL Documentation

```
namespace html = "http://www.w3.org/1999/xhtml/"

documentation = element documentation { inline?, doc-title?, html }

inline = attribute inline { ( "true" | "false" ) }
doc-title = element title { text }

html =  html-content*
html-content = (html-element | text | ref)
html-element = element html:* { html-attribute*, html-content* }
```

```
html-attribute = attribute * { text? }
```

C. DocBook module for RSDL Documentation

```
namespace docbook = "http://docbook.org/ns/docbook"

documentation = element documentation { inline?, doc-title?, docbook }

inline = attribute inline { ( "true" | "false" ) }
doc-title = element title { text }

docbook =  docbook-content*
docbook-content = (docbook-element | text | ref)
docbook-element = element docbook:* { docbook-attribute*, docbook-content* }
docbook-attribute = attribute * { text? }
```

D. An XSLT Stylesheet for RSDL

The following stylesheet converts a RSDL description to an HTML page.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:html="http://www.w3.
    xmlns:rsdl="http://identifiers.emc.com/rsdl">
  <xsl:output method="html" encoding ="utf-8" indent="no"/>
  <xsl:strip-space elements="*"/>

  <xsl:key name="status" match="//rsdl:status-codes/rsdl:status" use="@code" />

  <xsl:template match="/rsdl:service">
    <html>
      <head>
        <title><xsl:call-template name="title"/></title>
        <xsl:call-template name="style"/>
      </head>
      <body>
        <div class="outline index">
          <xsl:call-template name="index"/>
        </div>
        <div class="outline reference">
          <xsl:call-template name="reference"/>
        </div>
      </body>
    </html>
  </xsl:template>

  <xsl:template name="title">
    <xsl:value-of select="@name"/> REST Service
  </xsl:template>

  <xsl:template name="style">
```

```
  <style type="text/css">
    body { margin: 0; padding: 0 0 0 16em; }
    h1, h2 { color: Navy; }
    h3 { color: Blue; }
    table { border-collapse: collapse; margin-bottom: 1em; }
    th, td { border: 1px solid; padding: 0.35em; vertical-align: top; }
    th { color: White; background-color: CornflowerBlue; text-align: left; bo
    td { border-color: DarkBlue; }
    .outline { vertical-align: top; padding: 1em; }
    .index { position: fixed; top: 0; left: 0; width: 16em; height: 96%; overf
    .reference { height: 100%; overflow: auto; }
    div { overflow: scroll; }
    .homeResource {
      font-weight: bold;
      font-size: smaller;
      background-color: Green;
      color: Yellow;
      padding: 0.3em;
      margin-left: 1em;
      border-radius: 1em;
    }
    #hint {
      display: none;
      font-size: small;
      font-weight: normal;
      white-space: nowrap;
      background-color: LightYellow;
      color: DimGrey;
      border: 1px solid DimGrey;
      border-radius: 0.5em;
      padding: 0.5em;
      margin-right: 0.5em;
    }
    #full { color: Green; margin-right: 0.2em; }
    #full:hover ~ #hint { display: inline; }
    #no { color: Red; margin-right: 0.2em; }
    #no:hover ~ #hint { display: inline; }
    #partial { color: Coral; margin-right: 0.3em; }
    #partial:hover ~ #hint { display: inline; }
    .one-piece { white-space: nowrap; text-wrap: none; }
    .center { text-align: center; }
    .button {
      background-color: LightGrey;
      color: Black;
      font-size: smaller;
      border: 1px solid DarkGrey;
      border-radius: 0.35em;
      padding: 0.35em;
      text-decoration: none;
    }
    .header-suffix { font-size: small; }
  </style>
</xsl:template>

<xsl:template name="index">
  <xsl:call-template name="index-resources"/>
  <xsl:call-template name="index-media-types"/>
  <xsl:call-template name="index-link-relations"/>
```

```
  <xsl:call-template name="index-uri-parameters"/>
  <xsl:call-template name="index-custom-headers"/>
  <xsl:call-template name="index-status-codes"/>
  <xsl:call-template name="index-authentication-mechanisms"/>
</xsl:template>

<xsl:template name="index-resources">
  <h3>Resources</h3>
  <xsl:for-each select="//rsdl:resources/rsdl:resource">
    <xsl:sort select="@name"/>
    <a class="item">
      <xsl:attribute name="style">
        top: <xsl:value-of select="4 + position()"/>em;
      </xsl:attribute>
      <xsl:attribute name="href">#<xsl:value-of select="@id"/></xsl:attribute>
       <code><xsl:value-of select="@name"/></code>
    </a>
    <xsl:call-template name="home-resource"/>
    <br/>
  </xsl:for-each>
</xsl:template>

<xsl:template name="home-resource">
 <xsl:if test="@id = /rsdl:service/@home-resource">
   <span class="homeResource">Home resource</span>
 </xsl:if>
</xsl:template>

<xsl:template name="index-media-types">
  <h3>Media Types</h3>
  <xsl:for-each select="//rsdl:media-types/rsdl:media-type">
    <xsl:sort select="@name"/>
    <a class="item">
      <xsl:attribute name="style">
        top: <xsl:value-of select="4 + position()"/>em;
      </xsl:attribute>
      <xsl:attribute name="href">#<xsl:value-of select="@id"/></xsl:attribute>
       <code><xsl:value-of select="@name"/></code>
    </a>
    <br/>
  </xsl:for-each>
</xsl:template>

<xsl:template name="index-link-relations">
  <h3>Link Relations</h3>
  <xsl:for-each select="//rsdl:link-relations/rsdl:link-relation">
    <xsl:sort select="@name"/>
    <a class="item">
      <xsl:attribute name="style">
        top: <xsl:value-of select="4 + position()"/>em;
      </xsl:attribute>
      <xsl:attribute name="href">#<xsl:value-of select="@id"/></xsl:attribute>
       <code><xsl:value-of select="@name"/></code>
    </a>
    <br/>
  </xsl:for-each>
</xsl:template>
```

```
<xsl:template name="index-authentication-mechanisms">
  <xsl:if test="//rsdl:authentication/rsdl:uri-mechanism">
    <h3>Authentication Mechanisms</h3>
    <xsl:for-each select="//rsdl:authentication/rsdl:mechanism">
      <xsl:sort select="@name"/>
      <a class="item">
        <xsl:attribute name="style">
          top: <xsl:value-of select="4 + position()"/>em;
        </xsl:attribute>
        <xsl:attribute name="href">#<xsl:value-of select="@id"/></xsl:attribu
         <code><xsl:value-of select="@name"/></code>
      </a>
      <br/>
    </xsl:for-each>
  </xsl:if>
</xsl:template>

<xsl:template name="index-status-codes">
  <xsl:if test="//rsdl:status-codes/rsdl:status">
    <h3>Status Codes</h3>
    <xsl:for-each select="//rsdl:status-codes/rsdl:status">
      <xsl:sort select="@code"/>
      <a class="item">
        <xsl:attribute name="style">
          top: <xsl:value-of select="4 + position()"/>em;
        </xsl:attribute>
        <xsl:attribute name="href">#<xsl:value-of select="@id"/></xsl:attribu
         <code><xsl:value-of select="@code"/></code>
      </a>
      <br/>
    </xsl:for-each>
  </xsl:if>
</xsl:template>

<xsl:template name="index-custom-headers">
  <xsl:if test="//rsdl:custom-headers/rsdl:custom-header">
    <h3>Headers</h3>
    <xsl:for-each select="//rsdl:custom-headers/rsdl:custom-header">
      <xsl:sort select="@name"/>
      <a class="item">
        <xsl:attribute name="style">
          top: <xsl:value-of select="4 + position()"/>em;
        </xsl:attribute>
        <xsl:attribute name="href">#<xsl:value-of select="@id"/></xsl:attribu
         <code><xsl:value-of select="@name"/></code>
      </a>
      <br/>
    </xsl:for-each>
  </xsl:if>
</xsl:template>

<xsl:template name="index-uri-parameters">
  <xsl:if test="//rsdl:uri-parameters/rsdl:uri-parameter">
    <h3>URI Parameters</h3>
    <xsl:for-each select="//rsdl:uri-parameters/rsdl:uri-parameter">
      <xsl:sort select="@name"/>
      <a class="item">
        <xsl:attribute name="style">
```

```
            top: <xsl:value-of select="4 + position()"/>em;
          </xsl:attribute>
          <xsl:attribute name="href">#<xsl:value-of select="@id"/></xsl:attribu
           <code><xsl:value-of select="@name"/></code>
        </a>
        <br/>
      </xsl:for-each>
    </xsl:if>
  </xsl:template>

  <xsl:template name="reference">
    <h1><xsl:call-template name="title"/></h1>
    <xsl:call-template name="resources"/>
    <xsl:call-template name="media-types"/>
    <xsl:call-template name="link-relations"/>
    <xsl:call-template name="uri-parameters"/>
    <xsl:call-template name="custom-headers"/>
    <xsl:call-template name="status-codes"/>
    <xsl:call-template name="authentication-mechanisms"/>
  </xsl:template>

  <xsl:template name="resources">
    <h2>Resources</h2>
    <xsl:for-each select="//rsdl:resource">
      <xsl:sort select="@name"/>
      <hr/>
      <h3>
        <a>
          <xsl:attribute name="name"><xsl:value-of select="@id"/></xsl:attribute
        </a>
        <xsl:call-template name="implemented"/>
        <code><xsl:value-of select="@name"/></code>
        <xsl:call-template name="home-resource"/>
      </h3>
      <xsl:apply-templates select="rsdl:documentation"/>
      <xsl:call-template name="authentication"/>
      <xsl:apply-templates select="*[local-name() != 'methods' and local-name()
      <xsl:call-template name="methods"/>
    </xsl:for-each>
  </xsl:template>

  <xsl:template name="authentication">
    <xsl:choose>
      <xsl:when test="/service/@identity-provider-ref and @public = 'true'">
        <xsl:call-template name="no-authentication"/>
      </xsl:when>
      <xsl:when test="@identity-provider-ref">
        <xsl:call-template name="identity-provider">
          <xsl:with-param name="id" select="@identity-provider-ref"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:when test="/service/@identity-provider-ref">
        <xsl:variable name="id" select="/service/@identity-provider-ref"/>
        <h4>Authentication</h4>
        <p>
          <xsl:apply-templates select="//authentication/identity-provider[@id =
        </p>
      </xsl:when>
```

```
    <xsl:otherwise>
      <xsl:call-template name="no-authentication"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="no-authentication">
  <xsl:if test="//rsdl:authentication/rsdl:mechanism">
    <h4>Authentication</h4>
    <p>This resource requires no authentication.</p>
  </xsl:if>
</xsl:template>

<xsl:template name="identity-provider">
  <xsl:param name="id"/>
  <xsl:variable name="idp" select="//rsdl:authentication/rsdl:identity-provid
  <xsl:variable name="mechanismId" select="$idp/@mechanism-ref"/>
  <xsl:variable name="mechanism" select="//rsdl:authentication/rsdl:mechanism
  <h4>Authentication</h4>
  <table>
    <tr>
      <th>Mechanism</th>
      <th>Identity Provider</th>
    </tr>
    <tr>
      <td>
        <a>
          <xsl:attribute name="href">#<xsl:value-of select="$mechanismId"/></
          <xsl:apply-templates select="$mechanism/@name"/>
        </a>
      </td>
      <td>
        <xsl:apply-templates select="$idp/rsdl:documentation"/>
      </td>
    </tr>
  </table>
</xsl:template>

<xsl:template name="implemented">
  <xsl:choose>
    <xsl:when test="@status = 'full'">
      <span id="full">&#x2714;</span><span id="hint">Fully implemented</span>
    </xsl:when>
    <xsl:when test="@status = 'partial'">
      <span id="partial">?</span><span id="hint">Partially implemented</span>
    </xsl:when>
    <xsl:otherwise>
      <span id="no">&#x2718;</span><span id="hint">Not implemented</span>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="rsdl:documentation">
  <xsl:apply-templates select="text()|*"/>
</xsl:template>

<xsl:template match="rsdl:authentication">
  <xsl:variable name="mechanism" select="@mechanism-ref"/>
```

```
  <h4>Authentication</h4>
  <p>
    <a>
      <xsl:attribute name="href">#<xsl:value-of select="$mechanism"/></xsl:at
      <xsl:value-of select="//rsdl:authentication/rsdl:mechanism[@id = $mecha
    </a>.  <xsl:apply-templates select="*"/>
  </p>
</xsl:template>

<xsl:template match="*">
  <xsl:element name="{local-name()}">
    <xsl:apply-templates select="*|@*|text()"/>
  </xsl:element>
</xsl:template>

<xsl:template match="text()">
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="rsdl:ref">
  <xsl:choose>
    <xsl:when test="@resource and ancestor::rsdl:resource/@id = @resource">
      <code>
        <xsl:value-of select="@resource"/>
      </code>
    </xsl:when>
    <xsl:otherwise>
    <xsl:choose>
      <xsl:when test="@resource">
          <xsl:variable name="id" select="@resource"/>
          <xsl:call-template name="ref-by-id">
            <xsl:with-param name="id" select="$id"/>
            <xsl:with-param name="name" select="//rsdl:resources/rsdl:resourc
          </xsl:call-template>
      </xsl:when>
        <xsl:when test="@status-code">
          <xsl:variable name="id" select="@status-code"/>
          <xsl:call-template name="ref-by-id">
            <xsl:with-param name="id" select="$id"/>
            <xsl:with-param name="name" select="//rsdl:status-codes/rsdl:statu
          </xsl:call-template>
        </xsl:when>
        <xsl:when test="@uri-parameter">
          <xsl:variable name="id" select="@uri-parameter"/>
          <xsl:call-template name="ref-by-id">
            <xsl:with-param name="id" select="$id"/>
            <xsl:with-param name="name" select="//rsdl:uri-parameters/rsdl:uri
          </xsl:call-template>
        </xsl:when>
        <xsl:when test="@custom-header">
          <xsl:variable name="id" select="@custom-header"/>
          <xsl:call-template name="ref-by-id">
            <xsl:with-param name="id" select="$id"/>
            <xsl:with-param name="name" select="//rsdl:custom-headers/rsdl:cus
          </xsl:call-template>
        </xsl:when>
        <xsl:when test="@mechanism">
          <xsl:variable name="id" select="@mechanism"/>
```

```
            <xsl:call-template name="ref-by-id">
              <xsl:with-param name="id" select="$id"/>
              <xsl:with-param name="name" select="//rsdl:authentication/rsdl:me
            </xsl:call-template>
          </xsl:when>
          <xsl:when test="@media-type">
            <xsl:variable name="id" select="@media-type"/>
            <xsl:call-template name="ref-by-id">
              <xsl:with-param name="id" select="$id"/>
              <xsl:with-param name="name" select="//rsdl:media-types/rsdl:media
            </xsl:call-template>
          </xsl:when>
        <xsl:otherwise>
            <a>
            <xsl:attribute name="href">
              <xsl:value-of select="@uri"/>
            </xsl:attribute>
            <xsl:apply-templates select="*|text()"/>
            </a>
        </xsl:otherwise>
      </xsl:choose>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template name="ref-by-id">
    <xsl:param name="id"/>
    <xsl:param name="name"/>
    <a>
      <xsl:attribute name="href">#<xsl:value-of select="$id"/></xsl:attribute>
      <xsl:choose>
        <xsl:when test="text()">
          <xsl:apply-templates select="*|text()"/>
        </xsl:when>
        <xsl:otherwise>
          <code>
            <xsl:value-of select="$name"/>
          </code>
        </xsl:otherwise>
      </xsl:choose>
    </a>
  </xsl:template>

  <xsl:template name="methods">
    <xsl:if test="rsdl:methods/rsdl:method">
      <xsl:variable name="showStatus" select="rsdl:methods/rsdl:method/@status"
      <h4>Supported Methods</h4>
      <table>
        <tr>
          <xsl:if test="$showStatus">
            <th>Status</th>
          </xsl:if>
          <th>Name</th>
          <xsl:if test="rsdl:methods/rsdl:method/rsdl:request">
            <th>Request</th>
          </xsl:if>
          <xsl:if test="rsdl:methods/rsdl:method/rsdl:response">
            <th>Response</th>
```

```
        </xsl:if>
      </tr>
      <xsl:for-each select="rsdl:methods//rsdl:method">
        <tr>
          <xsl:if test="$showStatus">
            <td class="center">
              <xsl:call-template name="implemented"/>
            </td>
          </xsl:if>
          <td>
            <code><xsl:value-of select="@name"/></code>
          </td>
          <xsl:if test="../rsdl:method/rsdl:request">
            <td>
              <xsl:apply-templates select="rsdl:request/*"/>
            </td>
          </xsl:if>
          <xsl:if test="../rsdl:method/rsdl:response">
            <td>
              <xsl:apply-templates select="rsdl:response/*"/>
            </td>
          </xsl:if>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:if>
</xsl:template>

<xsl:template match="rsdl:status-codes">
  <em>Status code<xsl:if test="count(rsdl:status-code) &gt; 0">s</xsl:if>:</em
  <xsl:for-each select="rsdl:status-code">
    <xsl:variable name="statusId" select="@status-code-ref"/>
    <a>
      <xsl:attribute name="href">#<xsl:value-of select="$statusId"/></xsl:att
      <code><xsl:value-of select="//rsdl:status-codes/rsdl:status[@id = $stat
    </a>
    <xsl:if test="position() &lt; count(../rsdl:status-code)">,  </xsl:i
  </xsl:for-each>
  <br/>
</xsl:template>

<xsl:template match="rsdl:header">
  <xsl:variable name="id" select="@header-ref"/>
  <em>Header:</em> 
  <a>
    <xsl:attribute name="href">#<xsl:value-of select="$id"/></xsl:attribute>
    <code>
      <xsl:apply-templates select="//rsdl:custom-headers/rsdl:custom-header[@
    </code>
  </a>
  <br/>
</xsl:template>

<xsl:template match="rsdl:representation">
  <xsl:variable name="id" select="@media-type-ref"/>
  <xsl:if test="../rsdl:documentation">
    <br/>
  </xsl:if>
```

```
    <em>Media type:</em> <code>
    <a>
      <xsl:attribute name="href">#<xsl:value-of select="$id"/></xsl:attribute>
      <xsl:value-of select="//rsdl:media-types/rsdl:media-type[@id = $id]/@name
    </a>
    </code>
    <xsl:if test="@entity">, entity: <code><xsl:value-of select="@entity"/></co
    <xsl:if test="@entry">
      <xsl:variable name="resId" select="@entry"/>
      <xsl:text>, entry:</xsl:text> 
      <code>
        <a>
          <xsl:attribute name="href">#<xsl:value-of select="$resId"></xsl:value
          <xsl:value-of select="//rsdl:resources/rsdl:resource[@id = $resId]/@na
        </a>
      </code>
    </xsl:if>
    <xsl:if test="rsdl:documentation">
      <br/>    <xsl:apply-templates select="rsdl:documentati
    </xsl:if>
    <br/>
  </xsl:template>

  <xsl:template match="rsdl:links">
    <xsl:if test="rsdl:link">
      <h4>Links to other resources</h4>
      <xsl:variable name="showStatus" select="rsdl:link/@status"/>
     <table>
       <tr>
         <xsl:if test="$showStatus">
            <th>Status</th>
         </xsl:if>
         <th>Relation</th>
         <th>Resource</th>
          <xsl:if test="rsdl:link/rsdl:documentation">
           <th>Description</th>
          </xsl:if>
       </tr>
       <xsl:for-each select="rsdl:link">
         <xsl:sort select="@resource-ref"/>
          <xsl:variable name="resId" select="@resource-ref"/>
          <xsl:variable name="relId" select="@link-relation-ref"/>
         <tr>
            <xsl:if test="$showStatus">
              <td class="center">
                <xsl:call-template name="implemented"/>
              </td>
            </xsl:if>
           <td>
             <code>
                <a>
                  <xsl:attribute name="href">#<xsl:value-of select="$relId"></xs
                  <xsl:value-of select="//rsdl:link-relations/rsdl:link-relatio
                </a>
             </code>
           </td>
           <td>
             <code>
```

```
              <a>
                <xsl:attribute name="href">#<xsl:value-of select="$resId"></xs
                <xsl:value-of select="//rsdl:resources/rsdl:resource[@id = $re
              </a>
            </code>
          </td>
           <xsl:if test="../rsdl:link/rsdl:documentation">
            <td>
              <xsl:apply-templates select="rsdl:documentation/*|rsdl:documenta
            </td>
           </xsl:if>
         </tr>
       </xsl:for-each>
     </table>
   </xsl:if>
</xsl:template>

<xsl:template match="rsdl:location">
  <xsl:choose>
    <xsl:when test="../@id = /rsdl:service/@home-resource">
      <h4>Location</h4>
      <p>
        Reach this resource at <code><xsl:value-of select="@href"/></code>.
      </p>
    </xsl:when>
    <xsl:when test="@template and rsdl:var[@uri-parameter-ref]">
      <h4>URI Parameters</h4>
      <table>
        <tr>
          <th>Name</th>
          <th>Description</th>
        </tr>
        <xsl:for-each select="rsdl:var">
          <xsl:sort select="@name"/>
          <xsl:variable name="id" select="@uri-parameter-ref"/>
          <tr>
            <td><code><xsl:value-of select="@name"/></code></td>
            <td><xsl:apply-templates select="//rsdl:uri-parameters/rsdl:uri-pa
          </tr>
        </xsl:for-each>
      </table>
    </xsl:when>
  </xsl:choose>
</xsl:template>

<xsl:template name="authentication-mechanisms">
  <xsl:if test="//rsdl:authentication/rsdl:mechanism">
    <hr/>
    <h2>Authentication Mechanisms</h2>
    <xsl:for-each select="//rsdl:authentication/rsdl:mechanism">
      <xsl:sort select="@name"/>
      <h3>
        <a>
          <xsl:attribute name="name"><xsl:value-of select="@id"/></xsl:attribu
        </a>
        <code><xsl:value-of select="@name"/></code>
      </h3>
      <xsl:apply-templates select="rsdl:documentation"/>
```

```
      <xsl:for-each select="rsdl:scheme">
        <xsl:sort select="@name"/>
        <xsl:apply-templates select="." />
      </xsl:for-each>
    </xsl:for-each>
  </xsl:if>
</xsl:template>

<xsl:template match="rsdl:scheme">
  <h4>Scheme: <em><xsl:value-of select="@name"/></em></h4>
  <xsl:if test="rsdl:documentation">
    <p><xsl:apply-templates select="rsdl:documentation"/></p>
  </xsl:if>
  <xsl:if test="rsdl:parameter">
    <table>
      <tr>
        <th>Parameter</th>
        <th>Description</th>
      </tr>
      <xsl:for-each select="rsdl:parameter">
        <xsl:sort select="@name"/>
        <tr>
          <td><code><xsl:value-of select="@name"/></code></td>
          <td><xsl:apply-templates select="rsdl:documentation"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </xsl:if>
</xsl:template>

<xsl:template name="status-codes">
  <xsl:if test="//rsdl:status-codes/rsdl:status">
    <hr/>
    <h2>Status Codes</h2>
    <xsl:for-each select="//rsdl:status-codes/rsdl:status">
      <xsl:sort select="@code"/>
      <h3>
        <a>
          <xsl:attribute name="name"><xsl:value-of select="@id"/></xsl:attribu
        </a>
        <code><xsl:value-of select="@code"/></code>
      </h3>
      <xsl:apply-templates/>
    </xsl:for-each>
  </xsl:if>
</xsl:template>

<xsl:template name="custom-headers">
  <xsl:if test="//rsdl:custom-headers/rsdl:custom-header">
    <hr/>
    <h2>Headers</h2>
    <xsl:for-each select="//rsdl:custom-headers/rsdl:custom-header">
      <xsl:sort select="@name"/>
      <h3>
        <a>
          <xsl:attribute name="name"><xsl:value-of select="@id"/></xsl:attribu
        </a>
        <code><xsl:value-of select="@name"/></code>  <span class="he
```

```
      </h3>
      <xsl:apply-templates/>
    </xsl:for-each>
  </xsl:if>
</xsl:template>

<xsl:template name="media-types">
  <hr/>
  <h2>Media-types</h2>
  <xsl:for-each select="//rsdl:media-types/rsdl:media-type">
    <xsl:sort select="@name"/>
    <h3>
      <a>
        <xsl:attribute name="name"><xsl:value-of select="@id"/></xsl:attribut
      </a>
      <code><xsl:value-of select="@name"/></code>
    </h3>
    <xsl:apply-templates/>
  </xsl:for-each>
</xsl:template>

<xsl:template match="rsdl:description">
  <br/>
  <xsl:choose>
    <xsl:when test="@type = 'html'">
      <a>
        <xsl:attribute name="href">
          <xsl:value-of select="@href"/>
        </xsl:attribute>
        More information
      </a>
    </xsl:when>
    <xsl:when test="@type = 'sedola'">
      <a>
        <xsl:attribute name="href">
          <xsl:value-of select="@href"/>
        </xsl:attribute>
        Service registration
      </a>
    </xsl:when>
    <xsl:when test="@type = 'xsd'">
      <a>
        <xsl:attribute name="href">
          <xsl:value-of select="@href"/>
        </xsl:attribute>
        XML Schema
      </a>
    </xsl:when>
    <xsl:when test="@type = 'rnc'">
      <a>
        <xsl:attribute name="href">
          <xsl:value-of select="@href"/>
        </xsl:attribute>
        Relax NG Schema
      </a>
    </xsl:when>
  </xsl:choose>
</xsl:template>
```

```
  <xsl:template name="link-relations">
    <hr/>
    <h2>Link Relations</h2>
    <xsl:for-each select="//rsdl:link-relations/rsdl:link-relation">
      <xsl:sort select="@name"/>
      <h3>
        <a>
          <xsl:attribute name="name"><xsl:value-of select="@id"/></xsl:attribute
        </a>
        <code><xsl:value-of select="@name"/></code>
      </h3>
      <xsl:apply-templates/>
    </xsl:for-each>
  </xsl:template>

  <xsl:template name="uri-parameters">
    <xsl:if test="//rsdl:uri-parameters/rsdl:uri-parameter">
      <hr/>
      <h2>URI Parameters</h2>
      <xsl:for-each select="//rsdl:uri-parameters/rsdl:uri-parameter">
        <xsl:sort select="@name"/>
        <h3>
          <a>
            <xsl:attribute name="name"><xsl:value-of select="@id"/></xsl:attribu
          </a>
          <code><xsl:value-of select="@name"/></code>
        </h3>
        <xsl:apply-templates/>
      </xsl:for-each>
    </xsl:if>
  </xsl:template>

  <xsl:template match="rsdl:properties">
    <xsl:if test="rsdl:property">
      <h4>Properties</h4>
      <table>
        <tr>
          <th>Name</th>
          <th>Description</th>
        </tr>
        <xsl:for-each select="rsdl:property">
          <xsl:sort select="@name"/>
          <tr>
            <td><code><xsl:value-of select="@name"/></code></td>
            <td><xsl:apply-templates select="rsdl:documentation"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </xsl:if>
  </xsl:template>

</xsl:stylesheet>
```

E. RSDL Description for the Planets Service

The following example is taken from RESTful Web Services, chapter 5.

```
<?xml version="1.0" ?>
<service name="Maps"
  xmlns:html="http://www.w3.org/1999/xhtml/" xmlns="http://identifiers.emc.com/
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="htt
  >

  <documentation>
    This is an example from the book RESTful Web Services, chapter 5.
  </documentation>

  <start ref="res-planets"/>

  <media-types>
    <media-type id="med-xhtml" name="application/xhtml+xml">
      <documentation>
        We are defining an XHTML <html:em>microformat</html:em> by adding meani
        attribute to elements. For example, adding <html:code>class="planets"</
        <html:code>ul</html:code> element, we can turn a list into a list of pl
      </documentation>
      <description type="html" href="http://tools.ietf.org/html/rfc3236" />
    </media-type>
    <media-type id="med-png" name="image/png">
      <description type="html" href="http://www.iana.org/assignments/media-type
    </media-type>
  </media-types>

  <resources>
    <resource id="res-planets" name="planets">
      <location  uri="/"/>
      <links>
        <link link-relation-ref="rel-place" resource-ref="res-place"/>
      </links>
      <methods>
        <method name="GET">
          <response>
            <representation media-type-ref="med-xhtml" entity="html"/>
          </response>
        </method>
      </methods>
    </resource>

    <resource id="res-place" name="place">
      <location template="/{planet}/[{scoping-information}/][{place-name}]{?sho
        <var name="planet" uri-parameter-ref="par-planet"/>
        <var name="scoping-information" uri-parameter-ref="par-scoping-informat
        <var name="place-name" uri-parameter-ref="par-place-name"/>
        <var name="show" uri-parameter-ref="par-show"/>
      </location>
      <links>
        <link link-relation-ref="rel-map" resource-ref="res-map"/>
        <link link-relation-ref="rel-point" resource-ref="res-point"/>
        <link link-relation-ref="rel-place" resource-ref="res-place"/>
```

```
    </links>
    <methods>
      <method name="GET">
        <response>
          <representation media-type-ref="med-xhtml" entity="html"/>
        </response>
      </method>
    </methods>
  </resource>

  <resource id="res-point" name="point">
    <location template="/{planet}/{latitude},{longitude}">
      <var name="planet" uri-parameter-ref="par-planet"/>
      <var name="latitude" uri-parameter-ref="par-latitude"/>
      <var name="longitude" uri-parameter-ref="par-longitude"/>
    </location>
    <links>
      <link link-relation-ref="rel-place" resource-ref="res-place"/>
      <link link-relation-ref="rel-point" resource-ref="res-point"/>
    </links>
    <methods>
      <method name="GET">
        <response>
          <representation media-type-ref="med-xhtml" entity="html"/>
        </response>
      </method>
    </methods>
  </resource>

  <resource id="res-map" name="map">
    <location template="/{map-type}{scale}/{planet}/{latitude},{longitude}">
      <var name="map-type" uri-parameter-ref="par-map-type"/>
      <var name="scale" uri-parameter-ref="par-scale"/>
      <var name="planet" uri-parameter-ref="par-planet"/>
      <var name="latitude" uri-parameter-ref="par-latitude"/>
      <var name="longitude" uri-parameter-ref="par-longitude"/>
    </location>
    <links>
      <link link-relation-ref="rel-image" resource-ref="res-image"/>
      <link link-relation-ref="rel-map" resource-ref="res-map"/>
    </links>
    <methods>
      <method name="GET">
        <response>
          <representation media-type-ref="med-xhtml" entity="html"/>
        </response>
      </method>
    </methods>
  </resource>

  <resource id="res-image" name="image">
    <location template="/{map-type}{scale}/{planet}/images/{latitude},{longit
      <var name="map-type" uri-parameter-ref="par-map-type"/>
      <var name="scale" uri-parameter-ref="par-scale"/>
      <var name="planet" uri-parameter-ref="par-planet"/>
      <var name="latitude" uri-parameter-ref="par-latitude"/>
      <var name="longitude" uri-parameter-ref="par-longitude"/>
    </location>
```

```
    <methods>
      <method name="GET">
        <response>
          <representation media-type-ref="med-png"/>
        </response>
      </method>
    </methods>
  </resource>
</resources>

<link-relations>
  <link-relation id="rel-place" name="place">
    <documentation>
      The target resource is a related place. Links of this type are found by
      elements with <html:code>class="place"</html:code>. Additionally, you ca
      the <html:code>form</html:code> element with <html:code>id="searchPlace
    </documentation>
  </link-relation>
  <link-relation id="rel-point" name="point">
    <documentation>
      The target resource is a related point on a planet. Links of this type a
      <html:code>a</html:code> elements with different values for the <html:co
      like <html:code>coordinates</html:code>, <html:code>map_nav</html:code>
      and <html:code>zoom_out</html:code>.
    </documentation>
  </link-relation>
  <link-relation id="rel-map" name="map">
    <documentation>
      The target resource is a map related to the current resource. Links of t
      <html:code>a</html:code>
      elements with <html:code>class="map"</html:code>.
    </documentation>
  </link-relation>
  <link-relation id="rel-image" name="image">
    <documentation>
      The target resource is an image related to the current resource. Links o
      <html:code>img</html:code>
      elements with <html:code>class="map"</html:code>.
    </documentation>
  </link-relation>
</link-relations>

<uri-parameters>
  <uri-parameter id="par-planet" name="planet" datatype="string">
    <documentation>
      Human friendly name of a planet, like <html:code>Earth</html:code>.
    </documentation>
  </uri-parameter>
  <uri-parameter id="par-place-name" name="place-name" datatype="string">
    <documentation>
      Human friendly name of a place, like <html:code>Mount%20Rushmore</html:c
    </documentation>
  </uri-parameter>
  <uri-parameter id="par-scoping-information" name="scoping-information" data
    <documentation>
      A hierarchy of <ref uri-parameter="par-place-name">place names</ref> lik
      <html:code>/USA/New%20England/Maine/</html:code>.
    </documentation>
```

```
      </uri-parameter>
      <uri-parameter id="par-map-type" name="map-type" datatype="string">
        <documentation>
          The type of map, like <html:code>satellite</html:code>.
        </documentation>
      </uri-parameter>
      <uri-parameter id="par-scale" name="scale" datatype="string">
        <documentation>
          Dot followed by an integer, like <html:code>.1</html:code>. A bigger nu
        </documentation>
      </uri-parameter>
      <uri-parameter id="par-show" name="show" datatype="string">
        <documentation>
          Things to search for near a given place, like <html:code>diners</html:c
        </documentation>
      </uri-parameter>
      <uri-parameter id="par-latitude" name="latitude" datatype="float">
        <documentation>
          Latitude on a planet, like <html:code>24.9195</html:code>.
        </documentation>
      </uri-parameter>
      <uri-parameter id="par-longitude" name="longitude" datatype="float">
        <documentation>
          Longitude on a planet, like <html:code>17.821</html:code>.
        </documentation>
      </uri-parameter>
    </uri-parameters>
  </service>
```

Bibliography

[WADL] Marc Hadley, Sun Microsystems. *Web Application Description Language*, W3C Member Submission 31 August 2009. http://www.w3.org/Submission/wadl/.

[URI Templates] Joe Gregorio, Google; Roy Fielding, Adobe; Marc Hadley, MITRE; Mark Nottingham, Rackspace; David Orchard, Salesforce.com. *URI Template*, IETF RFC 6570, March 2012. http://tools.ietf.org/html/rfc6570

[JSON Home Documents] Mark Nottingham, Rackspace. *Home Documents for HTTP APIs*, May 8, 2013. http://www.ietf.org/id/draft-nottingham-json-home-03.txt

[XML Home Documents] Erik Wilde, EMC. *Home Documents for HTTP Services: XML Syntax*, June 11, 2013. http://www.ietf.org/id/draft-wilde-home-xml-01.txt

[Media Type Specifications and Registration Procedures] N. Freed, Oracle; J. Klensin; T. Hansen, AT&T Laboratories. *Media Type Specifications and Registration Procedures*, IETF RFC 6838, January 2013. http://tools.ietf.org/html/rfc6838

[To WADL or not to WADL] Bill Burke. *To WADL or not to WADL*, blog post, May 21, 2009. http://bill.burkecentral.com/2009/05/21/to-wadl-or-not-to-wadl/.

[REST APIs must be hypertext-driven] Roy Fielding. *REST APIs must be hypertext-driven*, blog post, Mon 20 Oct 2008. http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven/.

[Home Documents for HTTP APIs] *Home Documents for HTTP APIs*, http://tools.ietf.org/html/draft-nottingham-json-home-02. http://tools.ietf.org/html/draft-nottingham-json-home-02

[Problem Details for HTTP APIs] *Problem Details for HTTP APIs*, http://datatracker.ietf.org/doc/draft-nottingham-http-problem/. http://datatracker.ietf.org/doc/draft-nottingham-http-problem/

[XML Media Types] *XML Media Types*, IETF RFC 3023, MURATA Makoto (FAMILY Given), Simon St.Laurent, Daniel Kohn. http://tools.ietf.org/html/rfc3023

[Media Type Specifications and Registration Procedures] *Media Type Specifications and Registration Procedures*, IETF RFC 4288, Ned Freed, John C. Klensin. http://tools.ietf.org/html/rfc4288

[Additional Media Type Structured Syntax Suffixes] *Additional Media Type Structured Syntax Suffixes*, IETF RFC 5830, Tony Hansen, Alexey Melnikov. http://tools.ietf.org/html/rfc4288

[Does REST need a service description language] Aristotle Pagaltzis. *Does REST need a service description language?*, blog post, May 27, 2007. http://plasmasturm.org/log/460/.

[How RESTful is Your API?] Cory House. *How RESTful is your API?*, blog post, August 26, 2012. http://www.bitnative.com/2012/08/26/how-restful-is-your-api/.

[Richardson Maturity Model] Martin Fowler. *Richardson Maturity Model: steps toward the glory of REST*, blog post, 18 March 2010. http://martinfowler.com/articles/richardsonMaturityModel.html.

[What's Wrong with WADL?] Dare Obasanjo. *What's Wrong with WADL?*, blog post, June 4, 2007. http://www.25hoursaday.com/weblog/2007/06/04/WhatsWrongWithWADL.aspx

[REST in Practice] Jim Webber, Savas Parastatidis and Ian Robinson. *REST in Practice: Hypermedia and Systems Architecture*. O'Reilly Media; 1 edition (September 24, 2010). ISBN-13: 978-0596805821.

[RESTful Web Services] Leonard Richardson, Sam Ruby *RESTful Web Services*. O'Reilly Media; Dec 17, 2008f. ISBN-13: 978-0596554606.

[Sedola] Erik Wilde. *Service Documentation Language* https://github.com/dret/sedola/

[Architectural Styles and the Design of Network-based Software Architectures, PhD Dissertation Thesis.] Roy Thomas Fielding, *Architectural Styles and the Design of Network-based Software Architectures, PhD Dissertation Thesis*, University of California, Irvine © 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.