

# **Web Application Description Language (WADL)**

**Marc J. Hadley**

# Web Application Description Language (WADL)

**Marc J. Hadley**

SMLI TR-2006-153

March 2006

## **Abstract:**

This article describes the Web Application Description Language (WADL). An increasing number of Web-based enterprises (Google, Yahoo, Amazon, Flickr - to name but a few) are developing HTTP-based applications that provide access to their internal data using XML. Typically these applications are described using a combination of textual protocol descriptions combined with XML schema-based data format descriptions; WADL is designed to provide a machine processable protocol description format for use with such HTTP-based Web applications, especially those using XML.



Sun Labs  
16 Network Circle  
Menlo Park, CA 94025

**email address:**  
marc.hadley@sun.com

© 2006 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved. The SML Technical Report Series is published by Sun Microsystems Laboratories, of Sun Microsystems, Inc. Printed in U.S.A.

Unlimited copying without fee is permitted provided that the copies are not made nor distributed for direct commercial advantage, and credit to the source is given. Otherwise, no part of this work covered by copyright hereon may be reproduced in any form or by any means graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an information retrieval system, without the prior written permission of the copyright owner.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

TRADEMARKS Sun, Sun Microsystems, the Sun logo, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

For information regarding the SML Technical Report Series, contact Jeanie Treichel, Editor-in-Chief <jeanie.treichel@sun.com>. All technical reports are available online on our website, <http://research.sun.com/techrep/>.

# Web Application Description Language (WADL)

Marc J. Hadley, Sun Microsystems Inc.

March 2006

## 1 Introduction

This article describes the Web Application Description Language (WADL). WADL is designed to provide a machine process-able protocol description format for use with HTTP-based Web applications, especially those using XML.

### 1.1 Web Applications

For the purposes of this article, a Web application is defined as a dynamic HTTP-based application whose interactions are amenable to machine processing. While many existing Web sites are examples of dynamic HTTP-based applications, a large number of those require human cognitive function for successful non-brittle<sup>1</sup> use. Typically Web applications:

- Are based on existing Web architecture and infrastructure
- Are platform and programming language independent
- Promote re-use of the application beyond the browser
- Enable composition with other Web or desktop applications
- Require semantic clarity in content (representations) exchanged during their use

The latter requirement can be fulfilled by the use of XML either by defining a complete custom schema for the application domain or embedding a custom micro-format in an existing schema using its extensibility points. Given the above definition of a Web application, one can see that the following aspects of an application could be usefully described in a machine processable format:

**List of resources** Analogous to a site map showing the resources on offer.

---

<sup>1</sup>Brittle use, e.g., HTML page scraping, is generally always possible but less desirable in terms of maintenance.

**Relationships between resources** Describing the links between resources, both referential and causal.

**Methods that can be applied to each resource** The HTTP methods that can be applied to each resource, the expected inputs and outputs and their supported formats.

**Resource representation formats** The supported MIME types and any XML schemas in use.

## 1.2 Use Cases

The current state-of-the-art in Web application description is textual documentation plus one or more XML schemata. Whilst entirely adequate for human consumption, this level of description precludes the following use cases which require a more machine usable description format:

**Application Modelling and Visualization** Support for development of resource modelling tools for resource relationship and choreography analysis and manipulation.

**Code Generation** Automated generation of stub and skeleton code and code for manipulation of resource representations.

**Configuration** Configuration of client and server using a portable format.

It would also be useful to have a common foundation for individual applications and protocols to re-use and perhaps extend rather than each inventing a new description format.

## 1.3 Example WADL Description

The following listing shows an example of a WADL description for the Yahoo News Search[1] application.

```
1 <?xml version="1.0"?>
2 <application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://research.sun.com/wadl wadl.xsd"
4   xmlns:tns="urn:yahoo:yn"
5   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6   xmlns:yn="urn:yahoo:yn"
7   xmlns:ya="urn:yahoo:api"
8   xmlns="http://research.sun.com/wadl">
9   <grammars>
10    <include
11      href="NewsSearchResponse.xsd" />
12    <include
13      href="http://api.search.yahoo.com/Api/V1/error.xsd" />
14  </grammars>
15
16  <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
17    <resource uri="newsSearch">
```

```

18     <method href="#search"/>
19   </resource>
20 </resources>
21
22 <method name="GET" id="search">
23   <request>
24     <query_variable name="appid" type="xsd:string" required="true"/>
25     <query_variable name="query" type="xsd:string" required="true"/>
26     <query_variable name="type" type="xsd:string"/>
27     <query_variable name="results" type="xsd:int"/>
28     <query_variable name="start" type="xsd:int"/>
29     <query_variable name="sort" type="xsd:string"/>
30     <query_variable name="language" type="xsd:string"/>
31   </request>
32   <response>
33     <representation mediaType="application/xml" element="yn:ResultSet"/>
34     <fault id="SearchError" status="400" mediaType="application/xml"
35       element="ya:Error"/>
36   </response>
37 </method>
38 </application>

```

Lines 2–8 begin an application description and define the XML namespaces used elsewhere in the service description. Lines 9–14 define the XML grammars used by the service, in this case two W3C XML Schema files are included by reference. Lines 16–20 describe the Yahoo News Search Web resource and the HTTP methods it supports. Lines 22–37 describe the ‘search’ GET method: lines 23–31 describe the input; lines 32–36 describe the possible outputs.

## 2 Description Components

All WADL elements have the following XML namespace name:

- <http://research.sun.com/wadl>

This section describes each component of a WADL document in detail. Note that most elements are extensible either via additional attributes or child elements from another namespace. Unrecognized extensions may be ignored.

### 2.1 Application

The `application` element forms the root of a WADL description and contains the following:

1. An optional `grammars` element – see section 2.2.

2. An optional `resources` element – see section 2.3.
3. Zero or more of the following:
  - A `method` element – see section 2.5.
  - A `representation` element – see section 2.6.
  - A `fault` element – see section 2.7.

## 2.2 Grammars

The `grammars` element acts as a container for definitions of any XML structures exchanged during execution of the protocol described by the WADL document. Such definitions may be included inline or by reference using the `include` element (see section 2.2.1). No particular schema language is mandated; sections 3 and 4 describe use of RelaxNG and W3C XML Schema with WADL, respectively.

It is permissible to include multiple definitions of a particular XML structure: such definitions are assumed to be equivalent and consumers of a WADL description are free to choose amongst the alternatives or even combine them if they support that capability.

### 2.2.1 Include

The `include` element allows the definitions of one or more XML structures to be included by reference. The `href` attribute specifies a URI for the referenced definitions and is of type `xsd:anyURI`. Use of the `include` element is logically equivalent to in-lining the referenced document within the WADL `grammars` element.

## 2.3 Resources

The `resources` element acts as a container for the resources provided by the application. A `resources` element has a `base` attribute of type `xsd:anyURI` that specifies the base URI for each child resource identifier (see section 2.4). Child `resource` elements describe a single resource provided by the application.

### 2.4 Resource

The `resource` element describes a single resource provided by the Web application. A resource is identified by a URI and the `resources` parent element (see section 2.3) specifies the base URI for all child `resource` elements. A resource element has the following attributes:

**uri** An optional attribute of type `xsd:anyURI`. If present, it specifies the identifier of the resource as a static, relative URI whose base URI is given by the parent `resources` element. An implicit leading

'/' character is added to the value of the `uri` attribute. If the `uri` attribute is omitted, then the `resource` element **MUST** contain a child `path_variable` element (see section 2.4.1) that defines a parameterized relative URI whose base URI is given by the parent element.

A `resource` element contains the following child elements:

- An optional `path_variable` element; see section 2.4.1 that defines a parameterized relative URI for its parent `resource` element.
- Zero or more `method` (see section 2.5) elements, each of which describes the input to and output from an HTTP protocol method that can be applied to the resource.
- Zero or more `resource` elements that describe sub-resources.

The following example shows an extract from a Web application description that provides multiple resources:

```
1 <resources base="http://example.com/widgets">
2   <resource uri="stockreport">
3     ...
4   </resource>
5   <resource>
6     <path_variable name="widgetId" type="xsd:NMTOKEN"/>
7     ...
8   </resource>
9 </resources>
```

In the above example there are multiple resources:

- A single resource identified by a static URI: `http://example.com/widgets/stockreport`.
- Multiple resources identified by generative URIs: `http://example.com/widgets/widgetId`, where the `widgetId` component of the URI is replaced at runtime with the value of a runtime variable called `widgetId`.

In addition to being children of `resources` elements, `resource` elements may also be nested to an arbitrary level. In this case the child `resource` element describes a sub-resource and is identified relative to that of the parent `resource` element. The following example shows a WADL fragment that describes three resources:

```
1 <resources base="http://example.com/">
2   <resource uri="widgets">
3     ...
4     <resource uri="stockreport">
```

```

5      ...
6      <resource uri="">
7      ...
8      </resource>
9      ...
10     </resource>
11     ...
12     </resource>
13 </resources>

```

The resources described above are identified by three URIs:

- `http://example.com/widgets`
- `http://example.com/widgets/stockreport`
- `http://example.com/widgets/stockreport/`

### 2.4.1 Path Variable

The `path_variable` element is used to parameterize the identifier of its parent `resource` element (see section 2.4). A `path_variable` element has no defined child elements and has the following attributes:

**name** Specifies the name of the variable as an `xsd:NMTOKEN`. Required.

**type** Optionally specifies the type of the variable as an XML qualified name, defaults to `xsd:string`.

As is the case for the `uri` attribute of the `resource` element, an implicit leading `'/'` character is added to the value of the path variable.

## 2.5 Method

A `method` element describes the input to and output from an HTTP protocol method that may be applied to a resource. A `method` element has one of the two following combinations of attributes:

1. **name** Specifies the HTTP method used.  
**id** Specifies the identifier of the method.
2. **href** Allows a method defined elsewhere to be referenced by its `id` attribute, using a URI reference. This form of the method element is used to reduce the need for duplication when a method defined elsewhere also applies to the parent `resource` element. When the `href` attribute is present, the `method` element **MUST NOT** have any child content.

It is permissible to have multiple child `method` elements that have the same value of the `name` attribute for a given resource; such siblings represent distinct variations of the same HTTP method and will typically have different input data.

A `method` element has two child elements:

**request** Specifies the input to the method as a collection of variables and an optional resource representation – see section 2.5.1.

**response** Specifies the output of the method as a collection of alternate resource representations – see section 2.5.3.

### 2.5.1 Request

A `request` element describes the input to be included when applying an HTTP method to a resource. A `request` element has no attributes and may contain the following child elements in order:

1. Zero or more `representation` elements – see section 2.6. Note that use of `representation` elements is confined to HTTP methods that accept an entity body in the request (e.g., PUT or POST). Sibling `representation` elements represent logically equivalent alternatives, e.g., a particular resource might support multiple XML grammars for a particular request.
2. Zero or more `query_variable` elements – see section 2.5.2.

### 2.5.2 Query Variable

A `query_variable` element represents a URI query parameter as described in section 17.13 of HTML 4.01[2]. The runtime values of query variables are sent as URI query parameters when the HTTP method is invoked. A `query_variable` element has no defined child elements and has the following attributes:

**name** Specifies the name of the variable as an `xsd:NMTOKEN`. Required.

**type** Optionally specifies the type of the variable as an XML qualified name, defaults to `xsd:string`.

**required** Optionally specifies whether the variable is required to be present or not, defaults to false (not required).

**repeating** Optionally specifies whether the variable is single valued or may have multiple values, defaults to false (variable is single valued).

**fixed** Optionally specifies a fixed value for the variable.

The following example shows a resource with a generative URI that supports a single HTTP method with a single optional query variable:

```

1 <resources base="http://example.com/widgets">
2   <resource>
3     <path_variable name="widgetId" type="xsd:string"/>
4     <method id="GetDescription" name="GET">
5       <request>
6         <query_variable name="verbose" type="xsd:boolean"/>
7       </request>
8       <response>
9         ...
10      </response>
11    </method>
12  </resource>
13 </resources>

```

If the value of the `widgetId` variable is '1234567890' and the value of the `verbose` variable is 'true' then the URI on which the HTTP GET will be performed is:

```
http://example.com/widgets/1234567890?verbose=true
```

### 2.5.3 Response

A `response` element describes the output that results from performing an HTTP method on a resource. It contains zero or more of the following:

- A `representation` element - see section 2.6.
- A `fault` element - see section 2.7.

Each child `representation` element describes a resource state representation that may result from performing the method. Sibling `representation` elements indicate logically equivalent alternatives; normal HTTP mechanisms may be used to select a particular alternative. Each child `fault` element describes a fault condition that may occur – note that not all possible fault conditions are likely to be described and client applications should be prepared to handle the full range of possible HTTP error conditions.

## 2.6 Representation

A `representation` element describes a representation of a resource's state and can either be declared globally as a child of the `application` element, embedded locally as a child of a `request` or `response` element, or referenced externally. A `representation` element has one of the following two combinations of attributes:

1. **id** Specifies the identifier of the representation, required for globally defined representations, not allowed on locally embedded representations. Representations are identified by an XML ID and are referred to using a URI reference.

**mediaType** Specifies the media type of the representation.

**element** For XML-based representations, specifies the qualified name of the root element as described within the `grammars` section – see section 2.2.

2. **href** Allows a representation defined elsewhere to be referenced by its `id` attribute, using a URI reference. This form of the `representation` element may be used to reduce duplication when the same representation is used in multiple `request` or `response` elements. When the `href` attribute is present, the `representation` element **MUST NOT** have any child content.

In addition to the attributes listed above, a `representation` element can have zero or more child `representation_variable` elements, see section 2.6.1.

### 2.6.1 Representation Variable

A `representation_variable` element is used to parameterize its parent `representation` element. A `representation_variable` element has no defined child elements and has the following attributes:

**name** Specifies the name of the variable as an `xsd:NMTOKEN`. Required.

**type** Optionally specifies the type of the variable as an XML qualified name, defaults to `xsd:string`.

**path** Optionally specifies a path to the value of the variable.

**required** Optionally specifies whether the variable is required to be present or not, defaults to `false` (variable not required).

**repeating** Optionally specifies whether the variable is single valued or may have multiple values, defaults to `false` (variable is single valued).

**fixed** Optionally specifies a fixed value for the variable.

Representation variables can have one of two different functions depending on the media type of the representation:

1. Define the content of the representation. For `representation` elements with a `mediaType` attribute whose value is either `'application/x-www-form-urlencoded'` or `'multipart/form-data'` the representation variables define the content of the representation which is formatted according to the media type. The same may apply to other media types.

2. Provide a hint to processors about items of interest within a representation. For XML based representations, the representation variables can be used to identify items of interest with the XML. When used this way they are most useful for simple representations as in the example below where the information is contained within a few well-defined locations. The `path` attribute of a representation variable specifies the path to the value of the variable within the representation. For XML-based representations this is an XPath expression.

The following example shows an XML-based resource representation and two possible corresponding WADL representation elements:

```
1  <!-- XML-based representation of a widget -->
2  <w:widget xmlns:w="http://example.com/widgets">
3    <w:name>A Widget</w:name>
4    <w:description>A very useful gizmo.</w:description>
5    <w:price currency="USD">19.99</w:price>
6  </w:widget>
7
8  <!-- WADL fragment describing the widget representation without variables-->
9  <wadl:representation mediaType="application/xml" element="w:widget"/>
10
11 <!-- WADL fragment describing the widget representation with variables -->
12 <wadl:representation mediaType="application/xml" element="w:widget">
13   <wadl:representation_variable name="name"
14     type="xsd:string" path="/w:widget/w:name"/>
15   < wadl:representation_variable name="description"
16     type="xsd:string" path="/w:widget/w:description"/>
17   < wadl:representation_variable name="price"
18     type="xsd:decimal" path="/w:widget/w:price"/>
19   < wadl:representation_variable name="currency"
20     type="xsd:NMTOKEN" path="/w:widget/w:price/@currency"/>
21 </wadl:representation>
```

## 2.7 Fault

A `fault` element is similar to a `representation` element (see section 2.6) in structure but differs in that it denotes an error condition. A `fault` element has the same attributes as a `representation` element but may also have an additional `status` attribute that specifies a list of HTTP status codes associated with a particular error condition. Note that multiple `fault` elements may share one or more HTTP status codes: such elements may describe more granular fault conditions or may provide equivalent information in different formats.

### 2.7.1 Fault Variables

Fault variables are `representation_variable` elements that are direct children of a `fault` element. Fault variables perform the same function for `fault` elements that `representation` variables (see section

2.6.1) perform for `representation` elements.

## 2.8 Extensibility

Most WADL-defined elements are extensible using either elements or attributes from foreign namespaces. A WADL processor MAY ignore extensions that it does not understand and extension authors should design extensions with this in mind.

## 3 Use of RelaxNG with WADL

One or more legal RelaxNG schemas may be embedded within a WADL `grammars` element or may be included by reference using an `include` element. Multiple RelaxNG schemas may be combined within a single schema using the facilities provided by RelaxNG (e.g., `rng:include`). The default namespace for an included RelaxNG grammar is the default namespace of the WADL `grammars` element.

The `element` attribute of `representation` and `fault` elements refers to a corresponding RelaxNG element pattern using the XML qualified name of the element.

## 4 Use of W3C XML Schema with WADL

One or more legal W3C XML Schemas may be embedded within a WADL `grammars` element or may be included by reference using a `include` element. Multiple W3C XML Schemas may be combined within a single schema using the facilities provided by W3C XML Schema (e.g., `xsd:include`).

The `element` attribute of `representation` and `fault` elements refers to a corresponding W3C XML Schema global element declaration using the XML qualified name of the element.

## 5 WADL Media Type

WADL documents should be served using the `application/vnd.sun.wadl+xml` media type and use a `.wadl` filename extension. See the WADL media type registration[3] for full details.

## A Additional Examples

### A.1 Amazon Item Search

The following shows a WADL description of the Amazon item search service[4]:

```
1 <application xmlns="http://research.sun.com/wadl"
2   xmlns:aws="http://webservices.amazon.com/AWSECommerceService/2005-07-26"
3   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
4
5   <grammars>
6     <include href="AWSECommerceService.xsd"/>
7   </grammars>
8
9   <resources base="http://webservices.amazon.com/onca/">
10    <resource uri="xml">
11      <method href="#ItemSearch"/>
12    </resource>
13  </resources>
14
15  <method name="GET" id="ItemSearch">
16    <request>
17      <query_variable name="Service" fixed="AWSECommerceService"/>
18      <query_variable name="Version" fixed="2005-07-26"/>
19      <query_variable name="Operation" fixed="ItemSearch"/>
20      <query_variable name="SubscriptionId" type="xsd:string"
21        required="true"/>
22      <query_variable name="SearchIndex" type="aws:SearchIndexType"
23        required="true"/>
24      <query_variable name="Keywords" type="aws:KeywordList"
25        required="true"/>
26      <query_variable name="ResponseGroup" type="aws:ResponseGroupType"/>
27    </request>
28    <response>
29      <representation mediaType="text/xml"
30        element="aws:ItemSearchResponse"/>
31    </response>
32  </method>
33 </application>
```

Note the following:

- The method is attached to the resource as a reference to a globally defined method rather than being embedded directly as in the previous example. In this instance there is no need to do this beyond illustrating the capability but this is useful where one method can be applied to multiple resources.
- A number of the query variables are marked as fixed value. The Amazon API uses query parameters to identify services and operations within those services — use of the fixed attribute can be used to

allow description of multiple logical methods on the same resource. Without the ability to fix values in this way, the Amazon API would look like one single method with many parameters.

## A.2 Atom Publishing Protocol

The Atom publishing protocol[5] defines a set of methods to introspect, view and update entries in an Atom feed. The publishing protocol is bootstrapped using introspection on a known URI for a particular set of feeds. This process is described in the following WADL document:

```
1 <application xmlns="http://research.sun.com/wadl "  
2   xmlns:app="http://purl.org/atom/app#"  
3   xmlns:atom="http://www.w3.org/2005/Atom">  
4   <grammars>  
5     <include href="http://purl.org/atom/app.xsd" />  
6   </grammars>  
7  
8   <representation id="service" mediaType="application/atomserv+xml "  
9     element="app:service" />  
10  
11   <method name="GET" id="getService">  
12     <response>  
13       <representation href="#service" />  
14     </response>  
15   </method>  
16 </application>
```

Note the absence of a resources element. This is omitted since the document above defines a method that is detached from any specific Web site. WADL descriptions of a particular Web site can re-use the method definition for the resources it offers. The getService method response consists of an application/atomserv+xml document that describes the available feeds. An example of such is shown below:

```
1 <service xmlns="http://purl.org/atom/app#">  
2   <workspace title="Main Site" >  
3     <collection  
4       title="My Blog Entries"  
5       href="http://example.org/reilly/main" >  
6       <member-type>entry</member-type>  
7       <list-template>http://example.org/{index}</list-template>  
8     </collection>  
9     <collection  
10      title="Pictures"  
11      href="http://example.org/reilly/pic" >  
12      <member-type>media</member-type>  
13      <list-template>http://example.org/p/{index}</list-template>  
14    </collection>  
15  </workspace>  
16 </service>
```

Note the similarity between the Atom service document and WADL, both describe a set of resources and methods that may be applied to them. In the case of an Atom service document the applicable methods are implicit based on the member-type of a collection. An Atom service document also defines some additional metadata (the feed title) specific to the protocol domain. One could replicate the information in an Atom service document using WADL as follows.

The first step is to create a WADL document that contains all of the Atom protocol methods and associated representations. This only needs to be done once since the contents of this document can then be re-used by WADL documents specific to each site.

```
1 <application xmlns="http://research.sun.com/wadl "
2   xmlns:app="http://purl.org/atom/app#"
3   xmlns:atom="http://www.w3.org/2005/Atom">
4
5   <grammars>
6     <include href="http://purl.org/atom/app.xsd" />
7   </grammars>
8
9   <representation id="entry" mediaType="application/atom+xml "
10    element="atom:entry" />
11
12   <representation id="feed" mediaType="application/atom+xml "
13    element="atom:feed" />
14
15   <method name="GET" id="getFeed">
16     <response>
17       <representation href="#feed" />
18     </response>
19   </method>
20
21   <method name="POST" id="addEntryCollectionMember">
22     <request>
23       <representation href="#entry" />
24     </request>
25   </method>
26
27   <method name="POST" id="addGenericCollectionMember">
28     <request>
29       <representation href="#entry" />
30       <representation />
31     </request>
32   </method>
33
34   <method name="DELETE" id="deleteCollectionMember" />
35
36   <method name="GET" id="readEntryCollectionMember">
37     <response>
38       <representation href="#entry" />
39     </response>
40   </method>
41
42   <method name="GET" id="readGenericCollectionMember">
```

```

43     <response>
44         <representation href="#entry"/>
45     </representation />
46 </response>
47 </method>
48
49 <method name="PUT" id="updateEntryCollectionMember">
50     <request>
51         <representation href="#entry"/>
52     </request>
53     <response>
54         <representation href="#entry"/>
55     </response>
56 </method>
57
58 <method name="PUT" id="updateGenericCollectionMember">
59     <request>
60         <representation href="#entry"/>
61     </representation />
62 </request>
63     <response>
64         <representation href="#entry"/>
65     </representation />
66 </response>
67 </method>
68
69 </application>

```

Given the preceding document, one can create a WADL version of the prior Atom service document:

```

1 <application xmlns="http://research.sun.com/wadl "
2   xml:base="http://purl.org/atom/app.wadl "
3   xmlns:app="http://purl.org/atom/app#">
4
5   <resources base="http://example.org/">
6     <resource uri="reilly">
7       <resource uri="main" app:member-type="entry"
8         app:title="My Blog Entries">
9         <method href="#getFeed"/>
10        <method href="#addEntryCollectionMember"/>
11        <resource>
12          <path_variable name="range" type="app:indexRange"/>
13          <method href="#getFeed"/>
14        </resource>
15        <resource>
16          <path_variable name="entryId"/>
17          <method href="#readEntryCollectionMember"/>
18          <method href="#deleteCollectionMember"/>
19          <method href="#updateEntryCollectionMember"/>
20        </resource>
21      </resource>
22    </resource>

```

```
23     </resources>
24 </application>
```

Note the use of the `xml:base` attribute to allow use of relative URIs in method references. The above WADL document describes three resources:

**`http://example.org/reilly/main`** This resource supports HTTP GET to retrieve an Atom feed document and HTTP POST to add a new entry to the feed.

**`http://example.org/reilly/main/{range}`** Where `{range}` is a generative path segment that allows selection of a range of entries from the feed. This resource supports HTTP GET to retrieve an Atom feed document.

**`http://example.org/reilly/main/{entryId}`** Where `{entryId}` is a generative path segment that allows selections of a particular entry in the feed. This resource supports HTTP GET to retrieve an Atom entry document, HTTP PUT to replace an Atom entry in the feed, and HTTP DELETE to remove an entry from the feed.

The above document also includes some Atom-specific extension elements to provide the same metadata as the prior Atom service document.

## B RelaxNG Schema for WADL

```
1 namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
2 namespace local = ""
3 namespace wadl = "http://research.sun.com/wadl"
4
5 start =
6   element wadl:application {
7     grammars?,
8     resources?,
9     (method | representation | fault)*,
10    foreign-attribute,
11    foreign-element
12  }
13 grammars = element wadl:grammars { \include*, foreign-element }
14 resources =
15   element wadl:resources {
16     resource+,
17     attribute base { xsd:anyURI },
18     foreign-element
19   }
20 resource =
21   element wadl:resource {
22     (attribute uri { xsd:anyURI }
23      | path_variable),
24     (method | resource)+
25   }
26 method =
27   element wadl:method {
28     (attribute href { xsd:anyURI }
29      | (request?,
30         response?,
31         attribute id { xsd:token },
32         attribute name {
33           "DELETE" | "GET" | "HEAD" | "POST" | "PUT" | xsd:token " "
34         })),
35     foreign-element,
36     foreign-attribute
37   }
38 request =
39   element wadl:request {
40     representation*, query_variable*, foreign-attribute, foreign-element
41   }
42 response =
43   element wadl:response {
44     (representation | fault)*, foreign-attribute, foreign-element
45   }
46 representation =
47   element wadl:representation {
48     (attribute href { xsd:anyURI }
49      | (representation_variable*,
50         attribute id { xsd:token }?,
51         attribute element { xsd:QName }?,
```

```

52         attribute mediaType { text }?)),
53     foreign-attribute,
54     foreign-element
55 }
56 fault =
57     element wadl:fault {
58         (attribute href { xsd:anyURI }
59         | (representation_variable*,
60         attribute id { xsd:token }?,
61         attribute element { xsd:QName }?,
62         attribute mediaType { text }?,
63         attribute status {
64             list { xsd:int+ }
65         }?)),
66     foreign-attribute,
67     foreign-element
68 }
69 path_variable =
70     element wadl:path_variable {
71         attribute name { xsd:token },
72         attribute type { text }?,
73         foreign-element,
74         foreign-attribute
75     }
76 query_variable =
77     element wadl:query_variable {
78         attribute name { xsd:token },
79         attribute type { text }?,
80         attribute required { xsd:boolean }?,
81         attribute repeating { xsd:boolean }?,
82         attribute fixed { text }?,
83         foreign-element,
84         foreign-attribute
85     }
86 representation_variable =
87     element wadl:representation_variable {
88         attribute name { xsd:token },
89         attribute type { text }?,
90         attribute path { text }?,
91         attribute required { xsd:boolean }?,
92         attribute repeating { xsd:boolean }?,
93         attribute fixed { text }?,
94         foreign-element,
95         foreign-attribute
96     }
97 \include =
98     element wadl:include {
99         attribute href { xsd:anyURI },
100        foreign-attribute
101    }
102 foreign-attribute = attribute * - (wadl:* | local:*) { text }*
103 foreign-element =
104     element * - (wadl:* | local:*) {
105         (attribute * { text }

```

```
106     | text
107     | any-element)*
108   }*
109 any-element =
110   element * {
111     (attribute * { text }
112     | text
113     | any-element)*
114   }*
```

## C XML Schema for WADL

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3   targetNamespace="http://research.sun.com/wadl"
4   xmlns:tns="http://research.sun.com/wadl"
5   elementFormDefault="qualified">
6
7   <xs:element name="application">
8     <xs:complexType>
9       <xs:sequence>
10        <xs:element ref="tns:grammars" minOccurs="0"/>
11        <xs:element ref="tns:resources" minOccurs="0"/>
12        <xs:choice minOccurs="0" maxOccurs="unbounded">
13          <xs:element ref="tns:method"/>
14          <xs:element ref="tns:representation"/>
15          <xs:element ref="tns:fault"/>
16        </xs:choice>
17        <xs:any namespace="##other" processContents="lax"
18          minOccurs="0" maxOccurs="unbounded"/>
19      </xs:sequence>
20    </xs:complexType>
21  </xs:element>
22
23  <xs:element name="grammars">
24    <xs:complexType>
25      <xs:sequence>
26        <xs:element minOccurs="0" maxOccurs="unbounded"
27          ref="tns:include"/>
28        <xs:any namespace="##other" processContents="lax"
29          minOccurs="0" maxOccurs="unbounded"/>
30      </xs:sequence>
31    </xs:complexType>
32  </xs:element>
33
34  <xs:element name="resources">
35    <xs:complexType>
36      <xs:sequence>
37        <xs:element ref="tns:resource" maxOccurs="unbounded"/>
38        <xs:any namespace="##other" processContents="lax"
39          minOccurs="0" maxOccurs="unbounded"/>
40      </xs:sequence>
41      <xs:attribute name="base" type="xs:anyURI"/>
42    </xs:complexType>
43  </xs:element>
44
45  <xs:element name="resource">
46    <xs:complexType>
47      <xs:sequence>
48        <xs:element ref="tns:path_variable" minOccurs="0"/>
49        <xs:choice maxOccurs="unbounded">
50          <xs:element ref="tns:method"/>
51          <xs:element ref="tns:resource"/>
```

```

52         </xs:choice>
53         <xs:any minOccurs="0" maxOccurs="unbounded"
54             namespace="##other" processContents="lax"/>
55     </xs:sequence>
56     <xs:attribute name="uri" type="xs:anyURI"/>
57     <xs:anyAttribute namespace="##other" processContents="lax"/>
58 </xs:complexType>
59 </xs:element>
60
61 <xs:element name="method">
62     <xs:complexType>
63         <xs:sequence>
64             <xs:element ref="tns:request" minOccurs="0"/>
65             <xs:element ref="tns:response" minOccurs="0"/>
66             <xs:any namespace="##other" processContents="lax"
67                 minOccurs="0" maxOccurs="unbounded"/>
68         </xs:sequence>
69         <xs:attribute name="id" type="xs:ID"/>
70         <xs:attribute name="name" type="tns:Method"/>
71         <xs:attribute name="href" type="xs:anyURI"/>
72         <xs:anyAttribute namespace="##other" processContents="lax"/>
73     </xs:complexType>
74 </xs:element>
75
76 <xs:simpleType name="Method">
77     <xs:union memberTypes="tns:HTTPMethods xs:NMTOKEN"/>
78 </xs:simpleType>
79
80 <xs:simpleType name="HTTPMethods">
81     <xs:restriction base="xs:NMTOKEN">
82         <xs:enumeration value="GET"/>
83         <xs:enumeration value="POST"/>
84         <xs:enumeration value="PUT"/>
85         <xs:enumeration value="HEAD"/>
86         <xs:enumeration value="DELETE"/>
87     </xs:restriction>
88 </xs:simpleType>
89
90 <xs:element name="include">
91     <xs:complexType>
92         <xs:attribute name="href" type="xs:anyURI"/>
93         <xs:anyAttribute namespace="##other" processContents="lax"/>
94     </xs:complexType>
95 </xs:element>
96
97 <xs:element name="request">
98     <xs:complexType>
99         <xs:sequence>
100             <xs:element ref="tns:representation" minOccurs="0"
101                 maxOccurs="unbounded"/>
102             <xs:element ref="tns:query_variable" minOccurs="0"
103                 maxOccurs="unbounded"/>
104             <xs:any namespace="##other" processContents="lax"
105                 minOccurs="0" maxOccurs="unbounded"/>

```

```

106     </xs:sequence>
107     <xs:anyAttribute namespace="##other" processContents="lax"/>
108 </xs:complexType>
109 </xs:element>
110
111 <xs:element name="response">
112   <xs:complexType>
113     <xs:sequence>
114       <xs:choice minOccurs="0" maxOccurs="unbounded">
115         <xs:element ref="tns:representation"/>
116         <xs:element ref="tns:fault"/>
117       </xs:choice>
118       <xs:any namespace="##other" processContents="lax"
119         minOccurs="0" maxOccurs="unbounded"/>
120     </xs:sequence>
121     <xs:anyAttribute namespace="##other" processContents="lax"/>
122   </xs:complexType>
123 </xs:element>
124
125 <xs:element name="representation">
126   <xs:complexType>
127     <xs:sequence>
128       <xs:element ref="tns:representation_variable" minOccurs="0"
129         maxOccurs="unbounded"/>
130       <xs:any namespace="##other" processContents="lax"
131         minOccurs="0" maxOccurs="unbounded"/>
132     </xs:sequence>
133     <xs:attribute name="id" type="xs:ID"/>
134     <xs:attribute name="element" type="xs:QName"/>
135     <xs:attribute name="mediaType" type="xs:string"/>
136     <xs:attribute name="href" type="xs:anyURI"/>
137     <xs:anyAttribute namespace="##other" processContents="lax"/>
138   </xs:complexType>
139 </xs:element>
140
141 <xs:simpleType name="faultCodeList">
142   <xs:list itemType="xs:unsignedInt"/>
143 </xs:simpleType>
144
145 <xs:element name="fault">
146   <xs:complexType>
147     <xs:sequence>
148       <xs:element ref="tns:representation_variable" minOccurs="0"
149         maxOccurs="unbounded"/>
150       <xs:any namespace="##other" processContents="lax"
151         minOccurs="0" maxOccurs="unbounded"/>
152     </xs:sequence>
153     <xs:attribute name="id" type="xs:ID" use="required"/>
154     <xs:attribute name="element" type="xs:QName"/>
155     <xs:attribute name="status" type="tns:faultCodeList"/>
156     <xs:attribute name="mediaType" type="xs:string"/>
157     <xs:attribute name="href" type="xs:anyURI"/>
158     <xs:anyAttribute namespace="##other" processContents="lax"/>
159   </xs:complexType>

```

```

160 </xs:element>
161
162 <xs:element name="query_variable">
163   <xs:complexType>
164     <xs:sequence>
165       <xs:any namespace="##other" processContents="lax"
166         minOccurs="0" maxOccurs="unbounded"/>
167     </xs:sequence>
168     <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
169     <xs:attribute name="type" type="xs:QName" default="xs:string"/>
170     <xs:attribute name="required" type="xs:boolean"
171       default="false"/>
172     <xs:attribute name="repeating" type="xs:boolean"
173       default="false"/>
174     <xs:attribute name="fixed" type="xs:string"/>
175     <xs:anyAttribute namespace="##other" processContents="lax"/>
176   </xs:complexType>
177 </xs:element>
178
179 <xs:element name="path_variable">
180   <xs:complexType>
181     <xs:sequence>
182       <xs:any namespace="##other" processContents="lax"
183         minOccurs="0" maxOccurs="unbounded"/>
184     </xs:sequence>
185     <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
186     <xs:attribute name="type" type="xs:QName"
187       default="xs:string"/>
188     <xs:anyAttribute namespace="##other" processContents="lax"/>
189   </xs:complexType>
190 </xs:element>
191
192 <xs:element name="representation_variable">
193   <xs:complexType>
194     <xs:sequence>
195       <xs:any namespace="##other" processContents="lax"
196         minOccurs="0" maxOccurs="unbounded"/>
197     </xs:sequence>
198     <xs:attribute name="name" type="xs:NMTOKEN" use="required"/>
199     <xs:attribute name="type" type="xs:QName" default="xs:string"/>
200     <xs:attribute name="path" type="xs:string"/>
201     <xs:attribute name="required" type="xs:boolean"
202       default="false"/>
203     <xs:attribute name="repeating" type="xs:boolean"
204       default="false"/>
205     <xs:attribute name="fixed" type="xs:string"/>
206     <xs:anyAttribute namespace="##other" processContents="lax"/>
207   </xs:complexType>
208 </xs:element>
209
210 </xs:schema>

```

## References

- [1] Yahoo! Web APIs. Technical report, Yahoo!, 2005. See <http://developer.yahoo.net/>.
- [2] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification). Recommendation, W3C, December 1999. See <http://www.w3.org/TR/html4/>.
- [3] M. Hadley. The application/vnd.sun.wadl+xml Media Type. Media Type, IANA, March 2006. See <http://www.iana.org/assignments/media-types/application/vnd.sun.wadl+xml>.
- [4] Amazon.com. Amazon Web Services. Technical report, Amazon.com, 2005. See <http://www.amazon.com/>.
- [5] J.C. Gregorio and B. de hOra. The Atom Publishing Protocol. Internet Draft, IETF, January 2006. See <http://bitworking.org/projects/atom/draft-ietf-atompub-protocol-07.html>.

## Acknowledgments

Thanks to the members of the <http://lists.w3.org/Archives/Public/public-web-http-desc/> mailing list who provided useful feedback on several iterations of this specification. Mark Nottingham (Yahoo!) provided extensive feedback and helped structure the overall design.

## **About the Author**

Marc Hadley is an engineer in the Office of the CTO, Sun Microsystems. Marc represents Sun on the W3C XML Protocol, WS-Addressing and Web APIs working groups and is co-editor of the SOAP 1.2 and WS-Addressing 1.0 specifications. Previously he was co-specification lead for JAX-WS 2.0 (the Java API for Web Services) developed at the JCP and has also served as Sun's technical lead and alternate board member at the Web Services Interoperability Organization (WS-I).