LTER Web Services Recommendations

Published on *LTER Information Management* (http://im.lternet.edu)

LTER Web Services Recommendations

Mon, 09/20/2010 - 8:58am — mkortz

1 Introduction

This document is intended to serve as a guideline for developing web services within the LTER network information system. These guidelines apply to any services that will be exposed across sites, including the LTER Network Office. Web services only used internally within a site need not use these guides although doing so may allow for easier development and maintenance.

This document is divided into two main sections. The first section details recommendations for designing web services. This section is meant to be as implementation-agnostic as possible. The resources being delivered by the web service, the language in which it is written, the back-end storage, and other implementation details should not be affected by these recommendations.

The second section covers the documentation of new and existing web services. In order for the network to leverage web services, developers will need to know where to find the latest information on the services available and the methods for interacting with them.

2 Recommendations

The following are recommendations for exposing web services in the LTER network information system. Web services provide a architecture for sending and receiving information, not any particular end point or use of that information. As such, these recommendations cover only interactions between clients and the outward facing interface of the web service. Server-side processing and storage, and client processing after a completed request and response, are not addressed.

2.1 Rest Web Services

This recommendation document covers the use of the REST architecture for web services. REST architecture involves exchanging information as objects or resources, rather than as inputs and outputs to specific procedural calls. Recommendations for SOAP or other RPC-style web services may be addressed in other documents.

2.2 URL Syntax

Each web service in the NIS will have a well-defined URL syntax for accessing resources. The following guidelines aim to provide some continuity in the URL syntax between different web services. Doing so eases the task of developing against a wide set of services.
Web services syntax should be developed using the following guidelines:

- URLs should represent resources rather than actions.
- Each individual resource provided by a service should have a URL that is unique in the local network and on the World Wide Web.

- Resources that are arranged hierarchically should be separated with forward-slash characters.
- If a resource has only one representation, that representation should be indicated in the URL as a file extension.
- If a resource has more than one representation, the URL should provide a neutral identifier (i.e. no file extension).
- If the resource identified by the URL is an algorithm, non-identifying inputs to the algorithm should be placed in the query parameters (see URL Body vs. Query Parameters, below).
- List of resources should be treated as resources; optionally, lists can take query parameters for sorting and filtering members.

2.2.1 URL Body vs. Query Parameters

The body of a URL includes a protocol (HTTP or HTTPS for web services), a domain, and a path to a resource. The query parameters of a URL are a set of ampersand-separated key-value pairs, distinguished from the URL body using a question mark character.
For web services, information that identifies a resource should be contained entirely within the body of the URL. Query parameters should be reserved for non-identifying information, such as inputs to an algorithm or filters applied to a list. When in doubt, information should generally be moved from the query parameters to the URL body.

The following hypothetical URL is incorrectly formed, because identifying information about the resource (in this case, the name and formatting information) is included in the query parameter:

```
http://somewhere.lternet.edu/services/siteDB/getSite?name=CCE&format=xml
```

Instead the URL is better written as follows:

```
http://somewhere.lternet.edu/services/siteDB/sites/CCE.xml
```

The following are examples of correct usage of query parameters:

```
http://somewhere.lternet.edu/services/siteDB/sites?sortBy=dateAdded
http://somewhere.lternet.edu/services/siteDB/geoFinder?latitude=32.9&longitude=-120.3
```

2.2.2 Readability

URLs should be human-readable, provided this does not contradict the other guidelines given above. Hierarchical lists of resources should be evident in the URL as left-to-right, forward-slash-separated descriptive categories. If the service supports unique text names for resources, these named are preferred in URLs over numeric identifiers.

The following URL lacks readability:

```
http://somewhere.lternet.edu/services/cat_1/type_2/resource_3
```

Services should be written to allow more descriptive URLs:

`http://somewhere.lternet.edu/services/personnel/sitePersonnel/Mason_A_Kortz`

2.3 Requests

This subsection contains guidelines for requests made by clients (consumers) of web services. The guidelines address the proper use of certain HTTP/1.1 elements in constructing a request. The actual method of constructing the request may vary from client to client and does not alter the behavior of the service.

2.3.1 Methods

Each request to a web service must specify exactly one method, as per the HTTP/1.1 specification. Methods may be safe, idempotent, both, or neither.

- A **safe** method is a type of request that does not change the any information stored on the server.
- An **idempotent** method is a type of request where any number of identical requests has the same effect as a single request.

Eight methods are defined in the HTTP/1.1 specification. The following four methods are the most common for web service use:

- **GET:** Use when retrieving a resource. GET is a safe and idempotent method. This means a GET request should not be able to modify the information stored on the server and multiple identical GET requests should return the same response.
- **PUT:** Use when updating an existing resource. The new resource specified by the URL will be updated to match the representation sent in the request message. PUT is idempotent, as sending the same update multiple times should create no changes beyond the first.
- **POST:** Use when creating a new resource. The new resource will be created under the specified URL. The message body of the request should contain a representation of the resource to be created.
- **DELETE:** Use for any action that removes a resource. DELETE is idempotent; multiple calls to delete the same resource should not have any additional effect beyond the first.

2.3.2 Request Headers

A request sent to a web service may have any number of HTTP headers set. A complete list of standard headers can be found in the HTTP/1.1 specification. The following guidelines cover headers that are commonly important in web service requests:

Accept: Use to request a specific MIME type in the response. At least one acceptable content type must be set. Multiple types can be set as per the

HTTP/1.1 specification, but for machine-to-machine interaction it is recommended to request a single type.

Cache-Control: Use to specify whether the server (or any machine in the client-server chain) can use a cached copy. For most web service cases, this should be set to no-cache.

Content-Type: Use to specify the MIME type of the content in request message. If a request message is sent, this field must be set.

2.3.3 Request Message

The request message body should be used for PUT and POST requests to send a representation of a resource to be updated or created. The representation should be in the same format as the representations provided by the server. For example, if a server provides resources in XML representations, representations sent to the server in requests should also be in XML and should use the same schema.
The HTTP/1.1 specification does not prohibit message bodies for GET requests, however this use is discouraged.

2.4 Responses

This subsection contains guidelines for responses sent by servers (providers) of web services. The guidelines address the proper use of certain HTTP/1.1 elements in constructing a response, as well as the format of the response message body. Provided that the response follows these guidelines, the actual method of construction is irrelevant to the client. This section does not address the parsing, processing, or storage of a response after it has reached the client.

2.4.1 Response Codes

Every response from a web service must include a single HTTP response code. The full set of codes is defined in the HTTP/1.1 specification. The following guidelines address considerations specific to web services for certain codes:

- **200 OK:** Use for successful requests that do not create a new resource.
- **201 Created:** Use instead of 200 OK when a new resource has been created via a POST request.
- **400 Bad Request:** Use if the server cannot parse the request URL. If the request is parsable, a more specific code should be used.
- **404 Not Found:** Use if the request URL is parsable but does not specify an existing resource.
- **405 Method Not Allowed:** Use if the resource exists but request method (see above) is not supported by this service. In such a case, the Allow header should be set to inform the client of allowable methods.
- **406 Not Acceptable:** Use if the resource exists and the method is allowable, but the server cannot create a representation of the resource

that matches the client's Accept header. In such a case, the response message should contain a list of representations the server can create.
- **500 Internal Server Error:** Use when the client request is correct and supported, but a server-side error prevents successful processing.

2.4.2 Response Headers

A response sent by a web service may have any number of HTTP headers set. A complete list of standard headers can be found in the HTTP/1.1 specification. The following guidelines cover headers that are common in web service responses:

- **Allow:** Use in conjunction with a 405 Method Not Allowed response code to tell the client which methods are allowed.
- **Cache-Control:** Use to specify whether the client (or any machine in the client-server chain) can use a cached copy. For most web service cases, this should be set to no-cache.
- **Content-Encoding:** Use to specify the encoding of the response message. This should be used for very large response bodies that need to be compressed for bandwidth reasons.
- **Content-Type:** Use to specify the MIME type of the content in response message. If a response message is sent, this field must be set.
- **Last-Modified:** Use to specify the time at which the resource delivered in the response message was last modified. If a response message is sent, this field should be set.
- **Location:** Use in conjunction with a 201 Created response code to inform the client of the location of a newly created resource.

2.4.3 Response Message

For GET requests, the response message body should contain a representation of the requested resource. For PUT and POST requests, the message body should contain a representation of the updated or created resource. Responses to DELETE requests may contain a representation of the resource that was deleted, or be empty. In any case, the representation should be the best possible match for the Accept header of the request.

Services may provide resources in any format. Each response that includes a resource should have a Content-Type header specifying the format of the message body. When possible, the type should be drawn from the IANA registered MIME media types. The following content types are commonly used with web services:

- application/json
- application/xml or text/xml
- text/html
- text/csv
- text/plain

A set of recommendations specific to choosing a content type and schema for

LTER web services will be developed in future versions of this document.

HTML and certain XML schemas define mechanisms for representing links to other resources, including both static resources and those provided by web services. When possible, including a link to a related resource is preferable to including a copy of the related resource in the message body.

3 Documentation of Web Services

Each web service in the LTER network information system should be documented in a consistent manner. Human-readable documentation should contain at minimum the URL syntax for accessing resources, supported methods and their expected behaviors, and a description of the expected response format. A registry of LTER web services that tracks available services and documentation may be implemented in the future.

Machine-readable documentation for LTER web services may be supplied in the form of a web service description file. Web Service Description Language (WSDL) 2.0 includes improved support for REST web services. Other schema for describing web services have also been developed recently. Future versions of this document may include recommendations for creating machine-readable web service documentation.

- Web Services [1]

This material is based upon work supported by the National Science Foundation under Cooperative Agreement #DEB-0236154. Any opinions, findings, conclusions, or recommendations expressed in the material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
Please contact us with questions, comments, or for technical assistance regarding this web site.

Source URL: http://im.lternet.edu/news/committees/working_groups/webservices/recommendations

Links:
[1] http://im.lternet.edu/taxonomy/term/205