What is REST?

Representational State Transfer
The Representational State Transfer (REST) architectural style for services, first described by Ron Fielding ¹, was inspired by the development of Hypertext Transfer Protocol (HTTP) ² ³.

The main features of REST are:

- Client-Server: clients act as interfaces to server based services
- Stateless: each request must contain all the information required to interpret the request
- Cache: response data can be marked as cacheable or non-cacheable allowing use of caching at server, client or intermediate proxies.
- Uniform Interface
  - Identification of resources: unique identifiers (e.g. URLs or URIs) for specific resources
  - Manipulation of resources through representations: requests and responses are representations of objects (e.g. HTML or XML documents)
  - Self-descriptive messages: provision of meta-data describing the type of representation (e.g. MIME type)

These features are part of the HTTP design, thus HTTP can be considered a REST protocol and REST services are based on HTTP. Using URLs to identify resources, MIME types to identify types of data representation and using requests that contain all the state required to process the request.

HTTP
Currently REST services are typically implemented using HTTP, so it helps to know a little about the protocol and how it works.

Methods
HTTP defines four request methods which interact with URL:

- **GET**: retrieve the resource addressed by the URL
- **PUT**: send a resource to be placed at the address specified by the URL
- **POST**: send data to the URL and optionally receive a response
- **DELETE**: remove the resource addressed by the URL

REST Web Services currently tend to only use the **GET** and **POST** methods since these are well supported by proxies and firewalls, unlike the **PUT** and **DELETE** methods. Services which support **PUT** and **DELETE** usually provide alternative resources to support these methods via a **GET** or **POST** for compatibility.

Header Fields
In addition to the request method and URL optional header values can be included with the request. These are implemented as key/value pairs and can appear on both the request and the response. Headers which can be important in REST requests are:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>MIME types accepted by the client.</td>
</tr>
<tr>
<td>Content-Length</td>
<td>The size of the body content in bytes</td>
</tr>
<tr>
<td>Content-Type</td>
<td>MIME type of the body content.</td>
</tr>
</tbody>
</table>

A list of commonly used headers is available in Wikipedia.

Status Codes
The response from a HTTP server includes a status code, this indicates if the request was handled successfully, the service has moved or if an error occurred. The status code is also accompanied by a brief description for presentation to the user. Common examples include:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td>301</td>
<td>Moved Permanently</td>
</tr>
<tr>
<td>307</td>
<td>Temporary Redirect</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
</tr>
<tr>
<td>500</td>
<td>Internal Server Error</td>
</tr>
</tbody>
</table>

A complete list of the standard status codes can be found in Wikipedia. In a few cases the complete protocol is implemented by the Web Services server (e.g. DAS), which can mean the standard codes have a slightly different meaning. Check the documentation of the Web Service for details of status codes which are used by the service.
Data Representations

The data returned by the service can be in a variety of formats. Hints on how to handle the data returned can be found in the Content-Type header. If defined the Content-Type header contains the MIME type \(^{(4)}\) of the body content, which can be used to pass the data on to an appropriate viewer application or software module.

Some commonly used MIME types:

<table>
<thead>
<tr>
<th>MIME</th>
<th>Description</th>
<th>File extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/json</td>
<td>JavaScript Object Notation (JSON) (^{(5)}) document</td>
<td>.json</td>
</tr>
<tr>
<td>application/octet-stream</td>
<td>Binary data</td>
<td></td>
</tr>
<tr>
<td>application/xml</td>
<td>Extensible Markup Language (XML) (^{(6)}) document</td>
<td>.xml</td>
</tr>
<tr>
<td>chemical/x-pdb</td>
<td>Protein Data Bank (PDB) (^{(7)}) structure</td>
<td>.pdb</td>
</tr>
<tr>
<td>image/jpeg</td>
<td>JPEG (^{(8)}) image</td>
<td>.jpg or .jpeg</td>
</tr>
<tr>
<td>image/png</td>
<td>Portable Network Graphics (PNG) (^{(9)}) image</td>
<td>.png</td>
</tr>
<tr>
<td>text/html</td>
<td>Extensible HyperText Markup Language (XHTML) (^{(10)}) or HyperText Markup Language (HTML) (^{(11)}) document</td>
<td>.htm or .html</td>
</tr>
<tr>
<td>text/plain</td>
<td>Plain text</td>
<td>.txt</td>
</tr>
<tr>
<td>text/xml</td>
<td>Extensible Markup Language (XML) (^{(12)}) document</td>
<td>.xml</td>
</tr>
</tbody>
</table>

A complete list of official MIME types is available from [IANA](http://www.iana.org/). Some applications may use unofficial MIME types to allow data to be directed to external applications.

Error Handling

How errors are communicated to the client is one area where REST services are very diverse:

1. Web Services servers which directly implement the HTTP protocol (e.g. DAS) often use the HTTP status code mechanism to communicate error conditions.
2. Some services use additional HTTP headers to indicate error conditions
3. Others use error documents as part of the body content (e.g. dbfetch)

The exact mechanism used should be described in the documentation accompanying the service.

REST API Documentation

Due to the flexibility offered by REST individual services have their own behaviours and resource nomenclature. However there are some types of service that have well defined specifications which are widely implemented:

- AtomPub ([http://www.atomenabled.org/](http://www.atomenabled.org/)): an API for performing queries on categorized data and publishing data. Commonly used in blogging software, AtomPub is also used as a basis for APIs such as the Google Data API.
- Distributed Annotation System (DAS) \(^{(13)}\) ([http://www.biodas.org/](http://www.biodas.org/)): used to get sequence data and associated feature annotation.
- OpenSearch ([http://www.opensearch.org/](http://www.opensearch.org/)): an API for performing searches and getting search results. Commonly used for web browser search plug-ins. OpenSearch can also be used as an API if RSS or Atom feeds are used to describe the results.

These standards provide mechanisms to discover the capabilities of a service and how to invoke it at run time. This is useful when building clients since much of the work involved in interacting with a service can be handled by code generation, dynamic classes or client libraries. However it is necessary to know which type of service being accessed for the mechanisms provided to help.

A range of generic mechanisms exist for describing REST Web Service APIs:

- Norm’s Service Description Language (NSDL) \(^{(14)}\)
- RESTful Services Description Language (Resedel) \(^{(15)}\)
- Simple Message Exchange Descriptor (SMEX-D) \(^{(16)}\)
- Web Application Description Language (WADL) \(^{(17)}\)
- Web Description Language (WDL) \(^{(18)}\)
- Web Resource Description Language (WRDL) \(^{(19)}\)
- Web Services Description Language Version 2.0 (WSDL 2.0) \(^{(20)}\)

Currently WADL is seeing some adoption with the development of tools such as [RESTDescribe/RESTCompile](http://restful-labs.com/), the use of WADL in [Jersy](http://github.com/jersey) the reference implementation for [JAX-RS](http://jcp.org/en/jsr/detail?id=314) \(^{(21)}\), and the appearance of some services providing WADL descriptions, for example:

- [WSDbfetch (REST)](http://www.wsdfetch.org/)
WSDL 2.0 is also seeing use, for example: the SPARQL Protocol for RDF \(^{22}\) specification uses WSDL 2.0 to define the interface. Tools to manipulate WSDL 2.0 are appearing and Web Services tool-kits (e.g. Apache Axis2 \(^{23}\)) are starting to support WSDL 2.0.

---

4) RFC2046 - Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types - [http://www.faqs.org/rfcs/rfc2046.html](http://www.faqs.org/rfcs/rfc2046.html)
5) JavaScript Object Notation (JSON) - [http://www.json.org/](http://www.json.org/)
6) Extensible Markup Language (XML) - [http://www.w3.org/XML/](http://www.w3.org/XML/)
7) Protein Data Bank (PDB) - [http://www.wwpdb.org/](http://www.wwpdb.org/)
8) JPEG - [http://www.jpeg.org/jpeg/index.html](http://www.jpeg.org/jpeg/index.html)
9) Portable Network Graphics (PNG) - [http://www.w3.org/TR/PNG/](http://www.w3.org/TR/PNG/)
10) Extensible HyperText Markup Language (XHTML) - [http://www.w3.org/TR/html/](http://www.w3.org/TR/html/)
11) HyperText Markup Language (HTML) - [http://www.w3.org/TR/html4/](http://www.w3.org/TR/html4/)
13) Norm's Service Description Language (NSDL) - [http://norman.walsh.name/2005/03/12/nsdl](http://norman.walsh.name/2005/03/12/nsdl)
14) RESTful Services Description Language (Resedel) - [http://recycledknowledge.blogspot.com/2005/05/resedel.html](http://recycledknowledge.blogspot.com/2005/05/resedel.html)
15) Simple Message Exchange Descriptor (SMEX-D) - [http://www.ibrary.org/ongoing/When/200x/2005/05/03/SMEX-D](http://www.ibrary.org/ongoing/When/200x/2005/05/03/SMEX-D)
16) Web Application Description Language (WADL) - [https://wadl.dev.java.net/](https://wadl.dev.java.net/)
17) Web Description Language (WDL)
18) Web Resource Description Language (WRDL)
19) Web Services Description Language Version 2.0 (WSDL 2) - [http://www.w3.org/TR/wsdl20/](http://www.w3.org/TR/wsdl20/)
20) JAX-RS
21) SPARQL Protocol for RDF - [http://www.w3.org/TR/rdf-sparql-protocol/](http://www.w3.org/TR/rdf-sparql-protocol/)