THE BRITISH GEOLOGICAL SURVEY'S NEW GEOMAGNETIC DATA WEB SERVICE

E Dawson[1]*, J Lowndes[1] and P Reddy[2]

**[1]British Geological Survey, West Mains Road, Edinburgh EH3 9LA, United Kingdom*
Email:ewan@bgs.ac.uk
[2]School of Computing, Robert Gordon University, Schoolhill, Aberdeen AB10 1FR, United Kingdom

ABSTRACT

Increasing demand within the geomagnetism community for high quality real-time or near-real-time observatory data means there is a requirement for data producers to have a robust and scalable data processing infrastructure capable of delivering geomagnetic data products over the internet in a variety of formats. We describe a new software system, developed at BGS, which will allow access to our geomagnetic data products both within our organisation's intranet and over the internet. We demonstrate how the system is designed to afford easy access to the data from a wide range of software clients, and allows rapid development of software utilizing our observatory data.

Keywords: Web Services, REST, Geomagnetism, Data Visualisation, Java, Restlet

1 INTRODUCTION

The British Geological Survey (BGS) operates a network of observatories that continuously measure the geomagnetic field at eight locations around the world. Each of our geomagnetic observatories has one or more GDAS (Geomagnetic Data Acquisition Systems) instruments that sample the field at a rate of 1Hz. These data are transmitted to our data processing centre in Edinburgh over the internet in near-real-time. The data are stored on our Storage Area Network (SAN) in fixed-width format plain-text files in a hierarchical file system.

These continuous 1Hz time-series data comprise our raw base data, but our primary data product is a time-series of one-minute mean values for each observatory. As well as our primary data product, we also generate a number of derivative data products for use by the public, academics and industry. In addition, we also create other derivative data products for the purpose of detecting environmental interference and other issues in the data as part of our quality control procedures. These derivative data products are also stored in text files on our SAN.

Definitive data from geomagnetic observatories are available from third-party archives such as INTERMAGNET[1] and the World Data Centres for Geomagnetism[2]. However, due to the post-processing involved in deriving definitive geomagnetic field values from the observatory data (see Macmillan, 2007 for details), these definitive data only become available many months after recording. Some preliminary or quasi-definitive data (as defined in Peltier & Chulliat, 2010) are available much sooner from the BGS public website, although this is quite limited in terms of the range of data available, and is not easily accessible in a standard format.

In 2010, we started to look into using web services technology in order to facilitate easier access to our data products for both external and internal users. The result, which we describe here, is a web service providing timely access to a wide range of BGS geomagnetic observatory data products.

2 WEB SERVICE REQUIREMENTS

---

[1] http://www.intermagnet.org

[2] For example, http://www.wdc.bgs.ac.uk

The web service will be used by both internal and external users who require access to our geomagnetic observatory data products. It will also be used within BGS to provide data for visualisation tools to assist in observatory quality control procedures, and to display observatory magnetograms on our public website. From these use-cases we derived a number of requirements that the web service must satisfy. These are summarised below.

The web service should:

1. Allow the user to retrieve a time-series of geomagnetic field mean values and derivative data products using criteria such as: *observatory*, *interval* (start and end date of time-series) and *cadence* (second, minute or hour).
2. Allow the user to specify the format in which the data should be returned. Supported formats should include:
   - XML - for easy integration with third-party tools
   - JSON - to allow easy integration with browser-based web applications
   - IAGA-2002 - a plain-text data exchange format commonly used in the geomagnetism community (see McLean, 2011)
3. Respond to common queries in less than one second; the service must perform fast enough to be used by observatory operations staff in interactive data processing applications.
4. Be easily accessible by human users and programmatic clients alike.
5. Allow for easy integration with a wide range of commonly used tools, such as web-browsers, MATLAB, R, Excel, etc.
6. Allow for users to be authenticated, and restrict access to certain datasets depending on authorisation.

3 WEB SERVICE DESIGN

3.1 Data transfer protocol

There are a large number of protocols currently used in web services of one kind or another, each with its own pros, cons and common use cases. However, the only protocol which can be considered truly ubiquitous (and therefore meeting criteria requirement 5 above) is HTTP (Hypertext Transfer Protocol), the protocol which forms the basis of the World Wide Web. HTTP has well-defined semantics for retrieving and updating data over the internet (see Fielding *et al*, 1999 for details). It has client and server implementations available for almost all programming languages and operating systems, the most familiar examples being web browsers on the client side, and the Apache *httpd* web server on the server side.

3.2 Identification and representation of data

The HTTP protocol also offers a convenient way for us to identify our datasets, in the form of URLs. The various parameters which together identify the dataset are encoded within the URL itself, according to a defined schema. For example, if the parameters are *observatory*, *start date*, *end date* and *cadence*, a suitable URL schema would be:

```
http://geomag.bgs.ac.uk/webservice/obsdata/{observatory}/{cadence}/data?start={start date}&end={end date}
```

Using this URL schema, the one-minute-mean data for the first hour of January 31st, 2012 from Lerwick observatory would be identified by and retrieved using the URL:

```
http://geomag.bgs.ac.uk/webservice/obsdata/ler/minute/data?start=2012-01-31T00:00&end=2012-01-31T01:00
```

According to our requirements, we must also provide a way for the user to specify the format in which the data should be returned. The HTTP protocol defines a number of headers which may be sent as part of an HTTP message. One of these is the ‘Accept’ header, which allows the client to specify the preferred data format (or ‘media type’, in the language of the HTTP specification). Thus the user can specify the format by setting the ‘Accept’ header to the appropriate value, such as ‘application/xml’, ‘application/json’, or ‘application/x-iaga2002’. With some HTTP clients, for example web browsers, it is difficult to modify the HTTP request headers. To allow users to specify the required data format using such clients, we also allow the format to be specified as an additional URL parameter, for example by appending `&format=xml` to the URL.

3.3 Authentication and Authorisation

Some datasets, for example the derived data products used in our quality control procedures, should only be accessible to observatory operations staff. In order to satisfy this requirement, we need a way to authenticate (identify) the user, and authorise each request according to our data access rules. There are a number of authentication protocols in use on the web, but the simplest and most widely supported are the HTTP-Basic and HTTP-Digest authentication protocols that are part of the HTTP specification (see Franks *et al*, 1999). Both protocols require the client to send a username and password along with each request, which the web service can then use to establish the identity of the user. Once authenticated, the web service can then decide whether to allow or to deny access to the requested resource based upon the data access rules of the organisation.

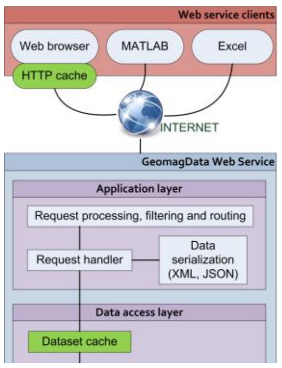

Figure 1. Architecture of the web service, showing the relationship between the major components. Some example clients are shown at the top.

4 IMPLEMENTATION

At BGS we already have the IT infrastructure and skills required to deploy web applications based on Java Servlet technology. Therefore when evaluating potential web service technologies we restricted ourselves to those that would be compatible with our existing infrastructure. Out of the various Java-based frameworks suitable for developing this kind of web service, we decided to use the *Restlet*[3] framework. *Restlet* is a popular open-source framework designed as an implementation of the “Representational State Transfer” (REST) style of information system architecture (for more details, see chapter 5 of Fielding, 2000). The design of our web service broadly conforms to the architectural constraints of REST[4], so the *Restlet* framework is a good fit. *Restlet* is mature, well documented, and has an active user community - all important considerations when deciding on the adoption of an open-source technology. The main benefit of using a framework such as this is that many of the components required to build the web service are provided by the framework, significantly reducing development time.

The architecture of the web service comprises two main layers (see Figure 1). The data access layer sits between the SAN data store and the web application layer. At the bottom, the data access layer is responsible for the low level interaction with the file system. It provides an interface to the web application layer, allowing access to the data without other parts of the system having to know anything about how the data are stored. This de-coupling of the application logic and the data access logic promotes resilience to change in the system; if we want to change our data storage system, for example by switching from using plain text files to using a relational database, then we need only provide a new implementation of the data access layer. No changes need be made to any other part of the system, and the change of data store will be transparent to the application logic.

The application layer, which is implemented using components provided by *Restlet*, encapsulates all of the application-level logic, such authentication and authorisation, parsing of request parameters, content-negotiation (determining in which format to return the data) and preparing the response.

The web service is packaged as a standard Java WAR package and deployed in an Apache *Tomcat* web application container. The web service runs on a 64-bit server with an 8-core CPU running at 2.8GHz and 16GB of RAM. This hardware allows us to service a very large number of concurrent requests; the software will also run satisfactorily in a less demanding environment on a standard consumer laptop computer with only 2GB of RAM.

[3] http://www.restlet.org

[4] One of the constraints that define the REST architectural style is the ‘hypertext constraint’: that is, all state transitions (including locating and navigating between resources) must be afforded via hypertext links. In our case, since we use a fixed URL schema to allow users to construct their own URLs rather than having them follow links from the web service root, our web service breaks the hypertext constraint.

4.1 Improving performance

There are a number of steps involved in a typical request-response cycle between the client and the web service. The client sends a request to the web service, which is then processed by the application layer. The application layer requests the data required to fulfil the request from the data access layer. These data may be spread across a number of files, each of which must be read into memory and parsed by the data access layer. The parsed data are then passed back to the application layer, where they must be re-serialized into the format requested by the client. Finally, the response containing the serialized data is then sent back across the network to the client.

Each of these steps adds to the amount of time taken to process the request. In particular, the reading and parsing of data from the disk, and the transmission of the data across the network are the most time-consuming steps, with the time taken increasing approximately linearly with the amount of data requested. In order to meet our performance requirement, we have implemented a number of strategies to minimize the latency of the typical request-response cycle. These optimisations are discussed briefly in the following sections.

4.1.1 Data access layer disk cache

The most time-consuming part of servicing a request for data is in reading and parsing the data from disk. To mitigate this we added caching functionality to the data access layer (see Figure 1). The cache stores frequently-requested data in RAM, so that subsequent requests for the same data may be serviced without having to access the disk at all (assuming the data on disk hasn't been modified in the interim). The caching technology used is Ehcache[5], a caching solution implemented entirely in Java. The cache is configured to use 8GB of the web server's RAM, which is enough to hold approximately 50,000 observatory daily time-series files. This form of caching decreases the time taken to respond to requests for commonly-accessed data by a factor of more than 100.

4.1.2 Compression of response

Even when data can be retrieved from the cache rather than disk, it must still be sent over the network to the client. The amount of data to be transferred can be quite large, especially since data formats such as XML are rather verbose. For example, three hours of one-second resolution data from one of our observatories translates to over 1MB of XML data.

We can reduce the amount of data to be transferred using a compressed encoding, such as GZIP. GZIP encoding is widely supported among HTTP clients, and is often handled completely transparently to the user. HTTP clients that can handle GZIP encoding indicate this capability to the web service by adding the 'accept-encoding: gzip' header to the request. The compressed response is typically 95% smaller, thus decreasing response transmission time by a factor of 20. Response compression incurs its own performance penalty though; it takes time for the server to compress the data, and for the client has to decompress it before it can be used. Although we did not measure the additional time taken for data compression/decompression, estimates based on user feedback indicate that response times decreased around ten-fold after compression was enabled on the server. This improvement was observed on a high-speed LAN; when accessing the web service over the Internet, the improvement observed should be greater still.

4.1.2 HTTP caching

The best optimisation is to avoid having to request the data in the first place. The HTTP protocol has a number of features which allow the web service to instruct the client as to which data may be cached (either locally, or by an intermediary server), and how to ask the web service if cached data is still fresh or should be replaced (see Fielding *et al*, 1999). HTTP caching is fully supported by the *Restlet* framework, and is widely supported in HTTP clients such as web browsers. This makes HTTP caching particularly useful in increasing the performance of interactive browser-based tools which make use of the web-service, such as the data visualization tool discussed later in Section 5.2.

[5] http://ehcache.org

By implementing disk caching, response compression, and HTTP caching, we have been able to increase the performance of the web service to meet the requirement that common requests are handled in less than a second. In fact, with these enhancements in place the majority of requests to the web service are handled in around a tenth of a second.

5 USING THE WEB SERVICE

5.1 Accessing the data via various HTTP-compatible clients

Because the web service operates using plain HTTP, any client capable of making an HTTP request can retrieve data using the web service. Since HTTP is one of the fundamental protocols of the internet, many modern software applications and tools are capable of retrieving data over HTTP. However in order to use the data, the application must not only be able to communicate with the web service, it must also understand the format in which the data are returned. Our web service exposes data in three formats: XML, which is one of the most commonly-used data exchange formats; JSON, which is rapidly becoming the de-facto data exchange format for web browser-based applications (see Crockford, 2006); and IAGA-2002, which is a standard data exchange format in the geomagnetism community (see McLean, 2011).

Examples of software packages which can easily consume data from our web-service are MATLAB, R, Mathematica and Microsoft Excel, all of which are capable of retrieving and parsing XML data from a URL.

In addition, almost all programming languages either have an HTTP client and XML parsing software included, or has them readily available as a third-party extension. This means it is easy to develop new software tools which make use of our latest geomagnetic observatory data in real-time.

5.2 Developing a browser-based data visualisation tool

In addition to providing external users with access to our data products, the web service also allows tools for internal use to be developed much more quickly than would be the case if the developer had to work directly with the data on the file system. The simple interface to the data provided by the web service frees the developer to concentrate on implementing the functionality of the tool.

We have developed a browser-based data visualization tool (see Figure 2), which uses the data provided by the web service to give a convenient overview of the data being recorded at each observatory and assist observatory operations staff in carrying out daily quality control checks on the data. The interface allows the user to browse plots of observatory data as well as various derived data products used to monitor the data quality. The tool was developed using the JavaScript language, and uses the *jQuery*[6] and *flot*[7] libraries for the user interface and magnetogram plotting, respectively.

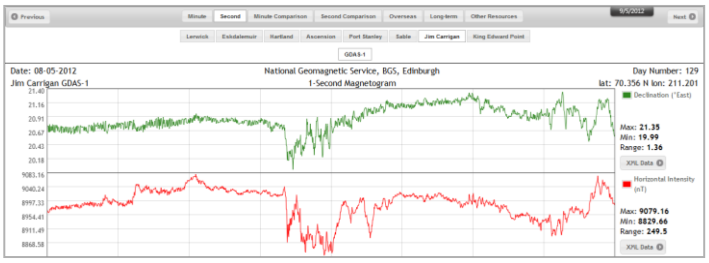

Figure 2. Screenshot of the data visualisation web application, plotting geomagnetic observatory data obtained in real-time from the web service.

The simple HTTP interface provided by the web service, coupled with the fact that the data can be delivered in the JavaScript-native JSON format makes building this kind of interactive web application relatively straightforward. Furthermore, because the client is not tightly coupled to the server – the client depending only on the semantics of the HTTP protocol and the URL schema we defined – we were able to make extensive modifications to the web service during its development with no impact to the client.

6 http://jquery.com

7 http://code.google.com/p/flot/

6 SUMMARY AND FUTURE DEVELOPEMENTS

Our new geomagnetic data web service provides a number of benefits to users, both within and outside the organization, who wish to make use of our observatory data products:

- Ease of access: software clients need only know how to access a URL and parse the response; no knowledge of how the data are stored in the repository is required.
- Reduction of code duplication: low level data access code is isolated in the web-service software – client software need not duplicate this code. This leads to faster and more reliable software development.
- Increased resilience to change: since the low-level data access details are abstracted away by the web service, changes to the way the data are stored (location, storage format, structure) need only be reflected in a single place – the data access layer of the web service – while clients using the data are unaffected.
- Interoperability: the web service can provide data in a variety of standard formats, reducing the need for client-side format translation and making it easier to integrate BGS geomagnetism data with existing software and systems.

Currently, our geomagnetism data web service is only available to users within BGS. However, we will make the web service publicly available in the near future, giving academics and the public unprecedented access to BGS geomagnetic data products.

7 ACKNOWLEDGEMENTS

This paper is published with the permission of the Director of the British Geological Survey (Natural Environment Research Council).

8 REFERENCES

Crockford, D. (2006) RFC 4627: *The application/json Media Type for JavaScript Object Notation (JSON).* Retrieved December 15, 2011 from the World Wide Web: http://www.ietf.org/rfc/rfc4627.txt

Fielding, R. T., Gettys, J., Mogul, J. C., Nielsen, H. F., Masinter, L., Leach, P. J. & Berners-Lee, T. (1999) *RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1.* Retrieved December 15, 2011 from the World Wide Web: http://tools.ietf.org/html/rfc261

Fielding, R. T. (2000) Architectural Styles and the Design of Network-based Software Architectures, Doctoral dissertation, University of California, Irvine, USA

Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Sink, E. & Stewart, L. (1999) *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication*. Retrieved December 15, 2011 from the World Wide Web: http://tools.ietf.org/html/rfc2617

Macmillan, S. (2007) Observatories : an overview. In: Gubbins, D.; Herrero-Bervera, E., (eds.) *Encyclopedia of Geomagnetism and Paleomagnetism.* Netherlands, Springer, 708-711, 1054pp. (Encyclopedia of Earth Sciences).

McLean, S. (2011) *IAGA2002 Data Exchange Format.* Retrieved December 15, 2011 from the World Wide Web: http://www.ngdc.noaa.gov/IAGA/vdat/iagaformat.html

Peltier, A. & Chulliat, A. (2010) *On the feasibility of promptly producing quasi-definitive magnetic observatory data.* Earth Planets Space, 62(2):e5-e8, doi:10.5047/eps.2010.02.002