Fermilab

GetDAQ Data Acquisition Web Service for ACNET Control System

Alexey Moroz <alexey.s.moroz@gmail.com>,

PARTI student, MIPT (http://phystech.edu/)

Supervisor: Andrey Petrov<apetrov@fnal.gov>,

AD/Controls, FNAL (http://www.fnal.gov/)

Abstract

GetDAQ application is a data acquisition web service for ACNET control system. It provides an opportunity for outer networks to read data from Fermilab's site-located devices.

Keywords: getdaq, web service, acnet, daq.

Contents

1 Introduction

1.1 ACNET Control System

ACNET is a uniform control system at Fermilab consisting of three main parts: frontends, middle layer and application layer. Experiments use a site-wide network of approximately 200,000 devices, each representing a single measurement point or a control item. Several devices could be united in one frontend, all of which in turn form a layer of the data acquisition (DAQ) frontends.

Middle layer servers are responsible for collecting data from the frontends and providing it to an upper application layer. Servers are usually referred to as data acquisition engines (DAE) and the total amount of the DAEs in Fermilab network is approximately 120 machines.

Finally, an application layer consisting of both console (legacy systems) and graphical (mostly Java-based) applications is used by operators and engineers for data analysis and system control.

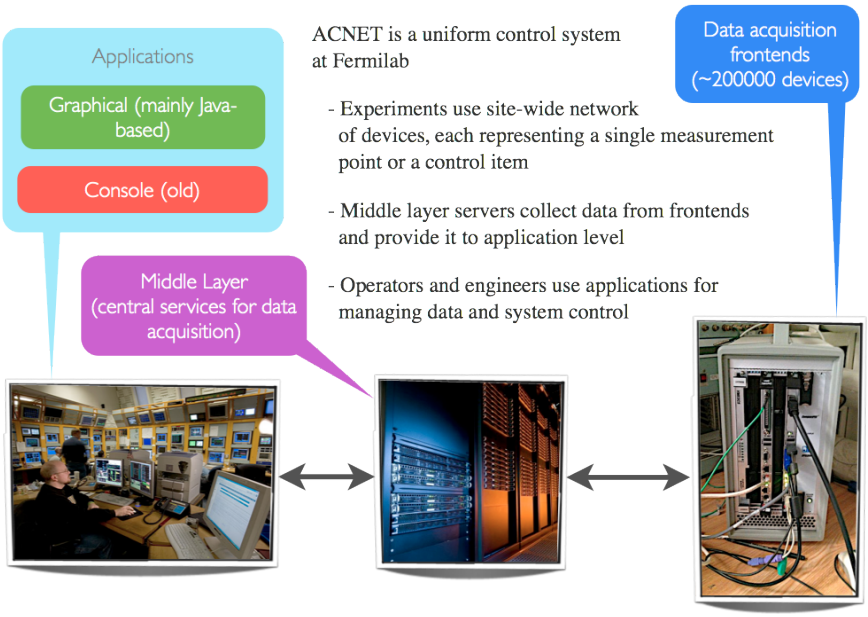

Figure 1: General overview of ACNET control system.

1.2 Problems Encountered

However, in its current state ACNET has some design problems, the most significant among them are:

- It was designed many years ago and was not supposed to provide data to the outside of the controls' network. Due to Tevatron's end of life, Fermilab will need to support a

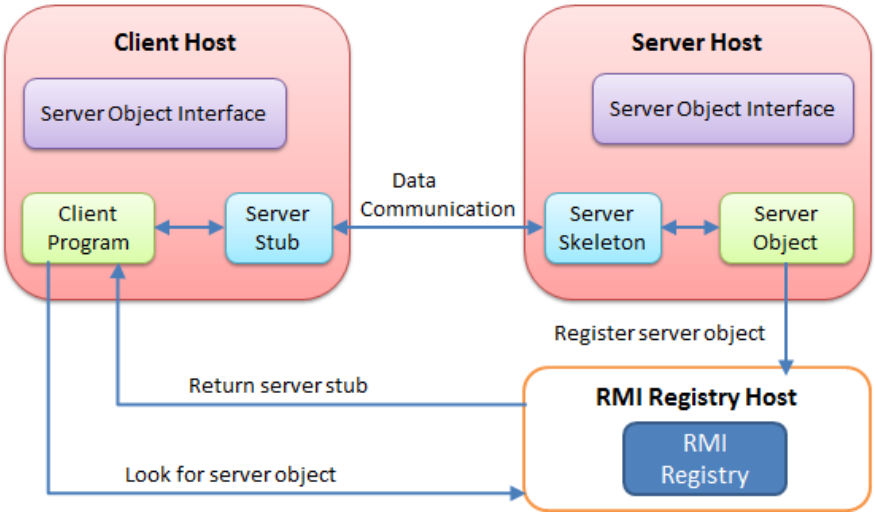

Figure 2: Overview of Java RMI technology.

number of geographically distributed research groups. All of them will need to maintain an access to data obtained on site. Moreover, the laboratory has no control over and influence on the research and software development processes in these groups.

- The current data acquisition system uses RMI (remote method invocation) technology, which is too complex and heavy-weight for the system in its current state.
- There is a need in a more simple way of acquiring data from the control system, because of a large number of platforms used by researchers, while RMI is Java-only binary protocol.

1.3 Decisions Made

Considering the problems stated above, a decision was made to introduce a more universal and flexible solution, particularly to use two different means for data acquisition from outer networks: an asynchronous AMPQ binary protocol and an HTTP-based request-response text protocol. On-site data acquisition still remains RMI-based, which is affordable in the current situation. Thus, the GetDAQ web service implements an HTTP interface, providing an opportunity of remote data acquisition.

In order to satisfy data requirements data, it is available in the 4 different formats: XML, HTML (human-readable), JSON (Javascript) and plaintext (csv), thereby allowing it to be accessible on all major software platforms. Furthermore, a caching system is introduced to prevent excessive load on DAE, since a request can hold a large number of devices queried. Also available is a statistics of entries and devices kept in the cache.

Finally, since a client can be both a human and a robot, data queries may be encapsulated into easily readable RESTful URL.

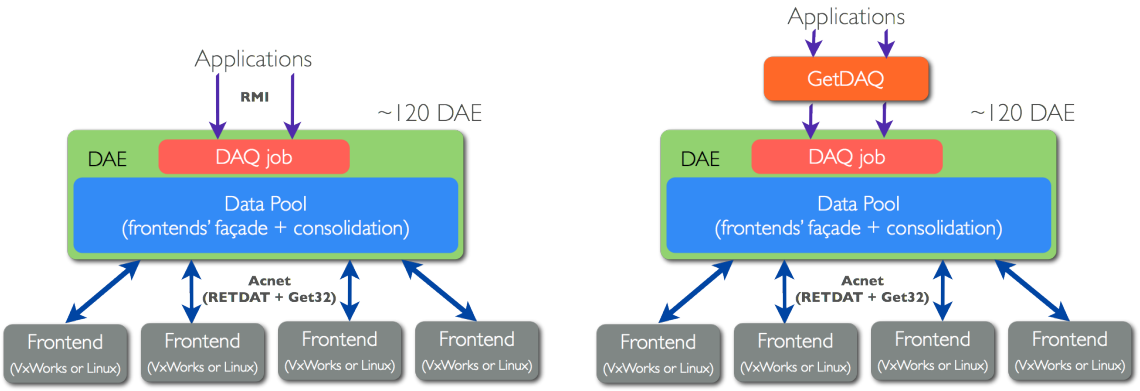

Figure 3: Controls Infrastructure with and without the GatDAQ web service.

2 General Overview

2.1 Controls Infrastructure

Shown on Figure 3 is an overview of the DAE infrastructure which existed before (Figure 3 (a)) and exists after (Figure 3 (b)) introducing the GetDAQ web service. DAEs are the custom application servers connected to each other into a load-balancing network. Applications query a DAE and create DAQ jobs. DAQ job is a responsible for data collection. DAE also runs a Data Pool which provides two major functions: frontends' facade and consolidation of requests. Communication between the DAE and the hardware frontends is held via a UDP-based Acnet protocol, which in turn is based on 2 low-level protocols: RETDAT (RETurn DATa) and Get32. Frontends usually run VxWorks or Linux as an OS. Thus, the web service is an additional layer between the user's application and the DAE, providing a mapping between a request-response pattern to a constantly running DAQ job thread.

2.2 Application Architecture

A more precise scheme of the application is shown on Figure 4. The client sends a request via HTTP to the servlet, which then calls a parser to form a binary object. Afterwards the request is passed to the cache manager, which in turn is responsible for creating new cache entries and starting the data acquisition. The cache entry starts communication with DAE and collects a reply data. Next, the servlet calls a marshaller and provides the reply data to it. Marshaller formats a reply and returns the reply's text representation to the servlet, which sends an HTTP reply to the client. The heart of the application is actually a caching system, which is responsible for mapping of the request-response pattern into a constantly running DAQ jobs.

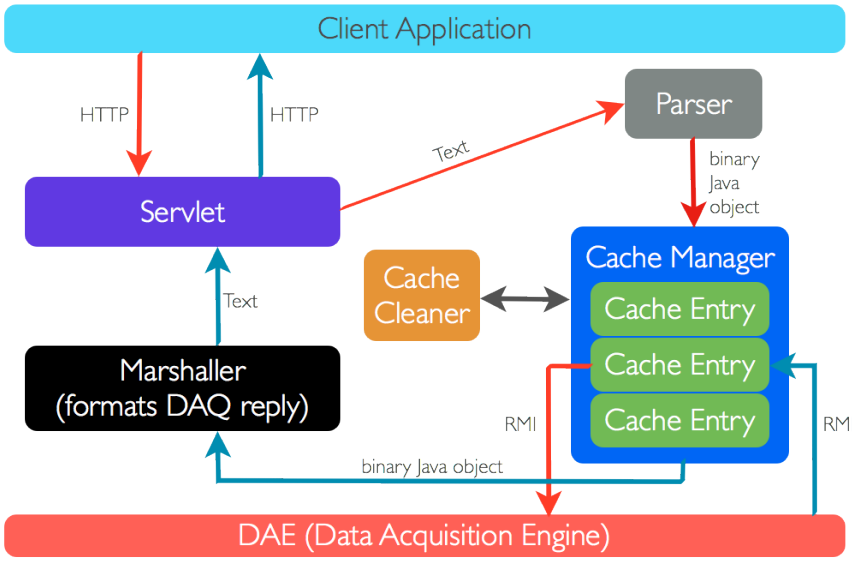

Figure 4: GetDAQ application architecture.

2.3 Caching System

The caching system (Figure 5) was introduced in order to prevent excessive load on DAE. Imagine, for example, a situation when several users query the same device. Starting a separate job for every user is not a good choice, yet the number of requests could be great. That is why the application uses a cache of DAQ entries, and every cache entry, roughly speaking, represents a separate DAQ job.

When a client requests data, the application searches for every requested device in its cache and, for those not found, starts a new DAQ thread. Due to this, some devices may return data later than other since right after its startup the DAQ job holds a "pending" status.

So, what happens with the jobs that are no longer required by anyone? They get stopped and deleted from the cache. This operation is maintained by a cache cleaner—a separate thread which works as a daemon and checks periodically if any entry in the cache was requested by anyone during last N minutes. If it was not, the cache cleaner removes the job from the cache list and stops the data acquisition.

While the cache system provides a good optimization, it has one drawback: If the application receives a large number of separate requests for a small number of non-overlapping devices, it will tend to create large number of DAQ jobs, therefore a job pool may become overflown, resulting in rejected job submissions. While this may not cause high server load, the utilization of resources in this case is far from optimal. Nevertheless, even if this case ever happens, it is unlikely that such a situation will last for a long time.

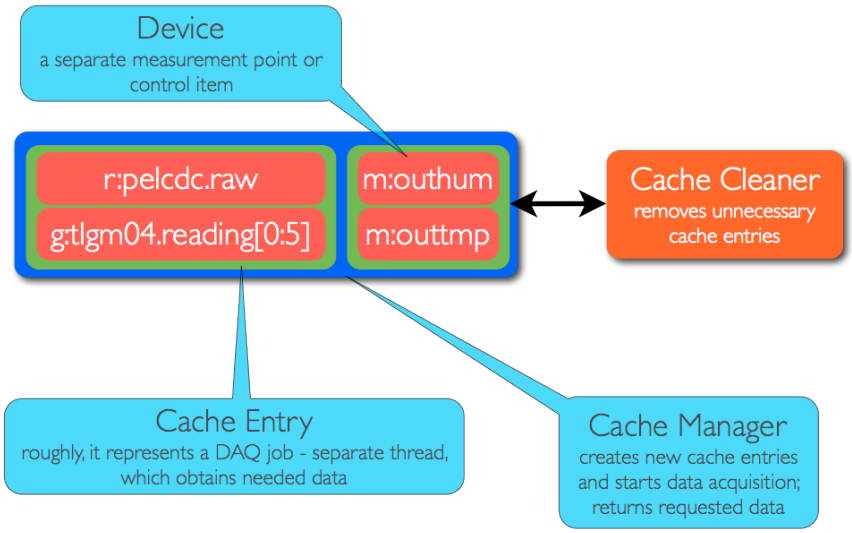

Figure 5: Cache system overview.

3 Technical Overview

3.1 Overall

The service is implemented by a Java servlet running on ACNET infrastructure. It accepts HTTP requests for data, parses the DAQ request and submits the request to a cache manager which is responsible for modifying the devices' cache. If the request contains devices which are not currently stored in the cache, the manager enqueues for startup a new DAQ job containing these devices. Right after the submittal, the device will have a "pending" status indicated by a status code «72 1». All replies of the devices are then marshalled to a representation specified by the client. By default, the service will return an XML document.

Due to a read-only nature of DAQ requests, no authentication mechanism was introduced. However, possible future extensions of the service may require an authorized access.

3.2 DAQ Request

The request string can be either a part of the URL or a POST parameter "request". There are two ways of writing the request: either by separating device names by brackets or semicolons. Device names must abide DRF2 specification.

Below are examples of using both methods. Note, that it should be specified explicitly if the request is bracket-separated. For the details, please refer to Section 3.3.

Example of the semicolon-separated request

```
m:outtmp;m:outhum;z:cache.reading;n:ccfoxy;i:beam
```

Device names MUST NOT contain any whitespaces. However, it is allowed to use spaces in between.

Also the user may use a bracket-separated form of the request, if it is more convenient:

Example of the bracket-separated request

```
(m:outtmp),(m:outhum),(z:cache.reading),(n:ccfoxy),(i:beam)
```

It is encouraged to use either commas or semicolons between brackets. Moreover, same constraints as above are applied on the device names.

3.3 HTTP Parameters

This section completely describes HTTP parameters which can be used for changing representation format.

Parameter	Type	Value	Default	Effect
`type`	string	`xml|html|plain|json`	`xml`	changes content-type (if no accept headers were sent).
`separator`	string	`semicolon|brackets`	`semicolon`	separator used: semicolon or brackets.
`iso-time`	boolean	`true|false`	`false`	If `true`, changes format of 'time' attribute to ISO8601.
`quiet`	boolean	`true|false`	`false`	If `true`, suppresses 'type' attribute for primitive types.
`include-drf2`	boolean	`true|false`	`false`	HTML-only. If `true`, includes canonical request.

3.4 GET Request via a RESTful URL

The user may query the service using URL in a format `http://<app-url>/job/<request>`, where `<app-url>` is a URL of the GetDAQ application (http://www-bd.fnal.gov/getdaq) and `<request>` is the request string specified in Section 3.2.

In order to choose a data representation format, the user may specify in the query string the desired string parameters described in Section 3.3.

3.5 POST Requests

To query a large set of data or just considering it a more convenient way, one may query devices using a POST method. The URL `<app-url>` for POST request is the same as in Section 3.4, however the URL should not contain a request string, instead of it request-body must provide `request` parameter containing the request string, along with the parameters from Section 3.3.

Note, that content-type `application/x-www-form-urlencoded` must be used in the request body.

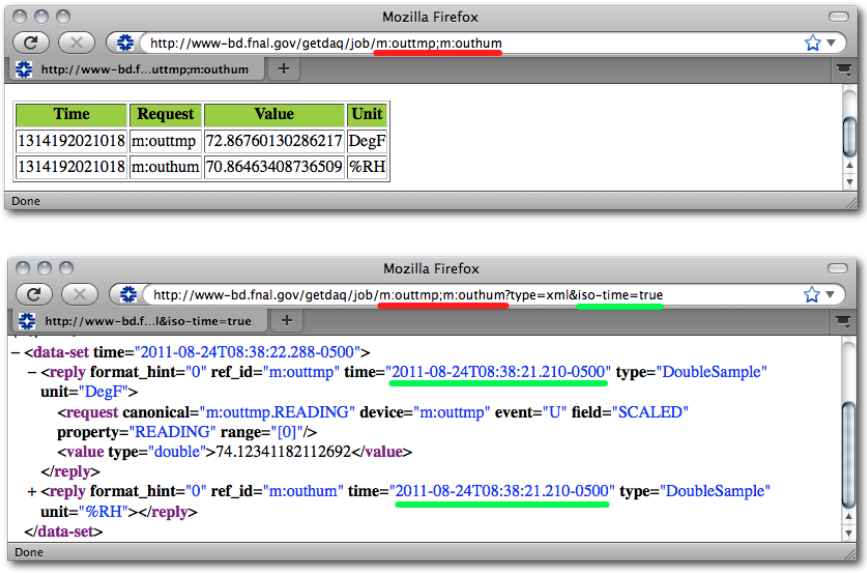

Figure 6: Examples of GET request.

3.6 Headers

The only thing which can be affected by HTTP headers is the content type. Instead of using a `type` parameter, the user may provide an `Accept` header. The content type which has the highest priority will be returned, `application/xml` is returned by default.

Note, that the `type` parameter has a strict priority over the headers, thus if both are present, content type specified in parameter will be returned.

Example of the header for which the response will be in HTML format

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

For more information, refer to RFC 2616.

3.7 HTTP Status Codes

This section describes HTTP status codes which may be generated by the web service.

Code	Status	Description
200	OK	Normal reply.
400	Bad Request	Error occured during request parsing. Probably, request string is invalid.
404	Not Found	Regular "Not Found" error.
405	Method Not Allowed	Method used is not allowed. The only methods allowed are GET and POST.
500	Internal Server Error	Severe error, refer to the server logs for details.
501	Not Implemented	Only HEAD method is not implemented.
503	Service Not Available	The service is encountering an overload problem. Try again later or change the application settings.

3.8 Reply Structure

The response XML format is completely described at http://www-bd.fnal.gov/issues/wiki/DAQDataXMLMarshalling. The HTML, JSON and CSV (plaintext) formats are the results of transformation of the initial XML. Please refer to Subsections 3.8.1, 3.8.2, 3.8.3, 3.8.4 for further information and examples.

3.8.1 XML

Below is an example of XML reply from the application.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Created by gov.fnal.controls.tools.daqdata.DAQDataXMLTransform-->
<data-set time="2011-09-20T18:24:59.567-0500"
 xmlns="http://www-bd.fnal.gov/2011/daqdata">
 <reply ref_id="m:outtmp"
  time="2011-09-20T18:24:58.799-0500"
  type="DoubleSample" unit="DegF">
   <request canonical="m:outtmp.READING"
    device="m:outtmp"
    event="U"
    field="SCALED"
    property="READING"
    range="[0]"/>
   <value type="double">52.07696050047875</value>
 </reply>
</data-set>
```

3.8.2 HTML

The HTML representation is a result of an XSLT transformation of the XML document. It contains 4 columns if there is no `include-drf2` parameter or the parameter is set to

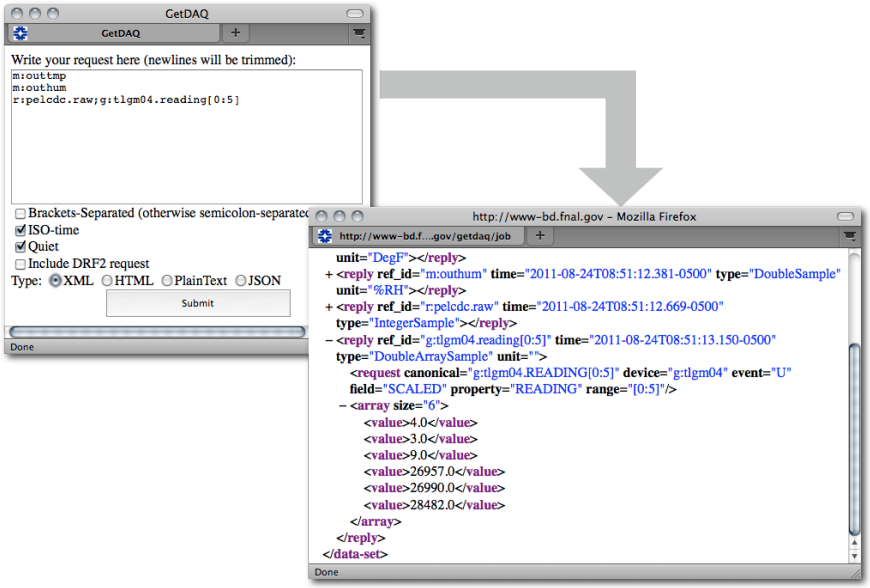

Figure 7: Example of XML reply.

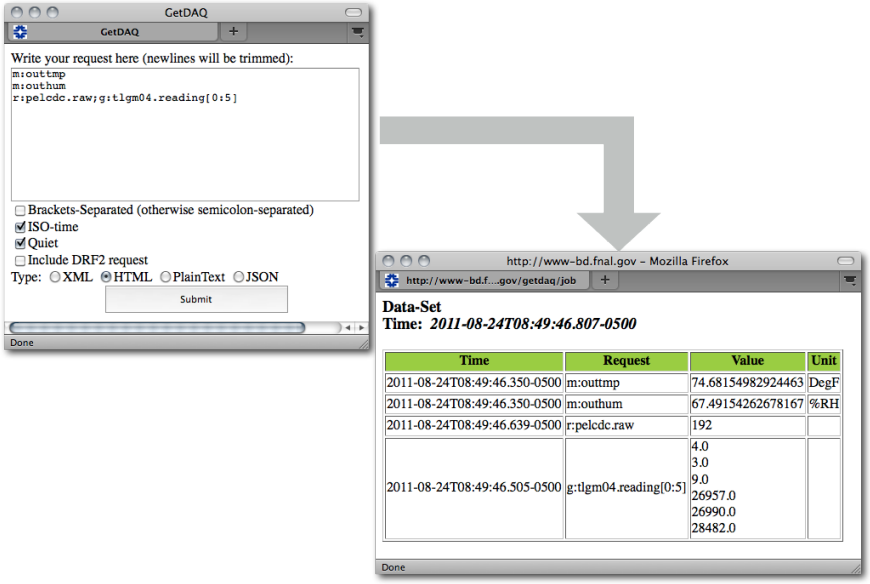

Figure 8: Example of HTML reply.

false, and 5 columns otherwise. The column names are "Time" (shows time when data was collected), "Request", "Canonical" (optional), "Value" and "Unit".

3.8.3 JSON

The JSON representation is a Javascript object containing all fields and attributes of the XML representation (also preserving the structure). For more information on the Javascript

object model, refer to http://json.org/.

Below is an example of the JSON object returned in a reply.

```
{"data-set": {
  "time": "2011-09-20T18:27:39.438-0500",
  "reply": {
    "unit": "DegF",
    "time": "2011-09-20T18:27:38.807-0500",
    "request": {
      "field": "SCALED",
      "range": "[0]",
      "event": "U",
      "device": "m:outtmp",
      "property": "READING",
      "canonical": "m:outtmp.READING"
    },
    "value": {
      "content": 51.72812424540521,
      "type": "double"
    },
    "ref_id": "m:outtmp",
    "type": "DoubleSample"
  },
  "xmlns": "http://www-bd.fnal.gov/2011/daqdata"
}}
```

3.8.4 Plaintext

The plaintext representation is a simple file format storing tabular data where semicolons are used as a delimiters between columns. It contains the following fields: “Time”, “Request”, “Canonical”, “Device”, “Event”, “Field”, “Property”, “Range”, “Type”, “Value”, “Unit”.

Below is an example of a plaintext reply.

```
Time; Request; Canonical; Device; Event; Field; Property; Range; Type; Value; Unit
1316496341801; m:outtmp; m:outtmp.READING; m:outtmp; U; SCALED; READING; [0]; double; 51.93742599844933; DegF
```

3.9 Statistics Page

For the purpose of statistics gathering, there exists a status page, which returns an XML document with information about entries and devices currently kept in cache.

Page URL is http://www-bd.fnal.gov/getdaq/status.

Below is an example of a status reply.

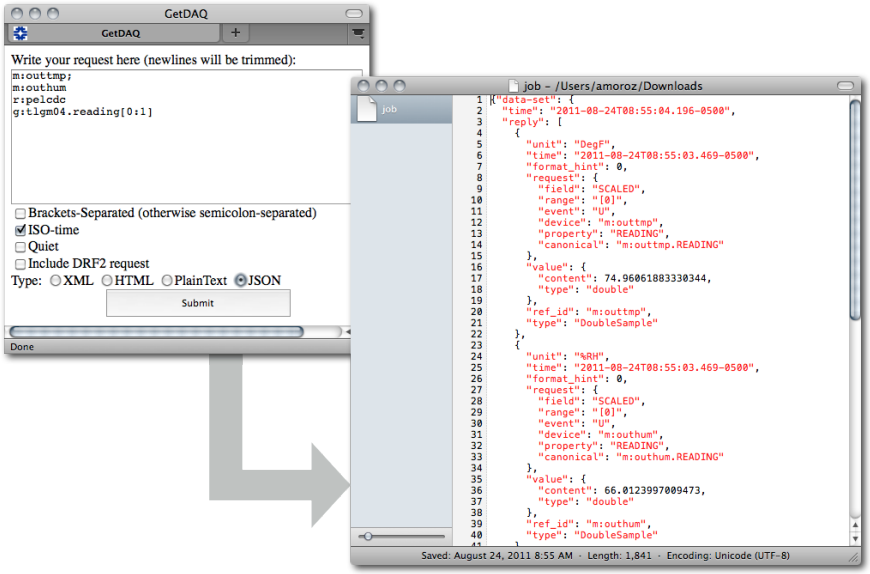

Figure 9: Example of JSON reply.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Created by gov.fnal.controls.tools.daqdata.DAQDataXMLTransform-->
<cache currentTime="2011-09-20T18:55:23.381-0500"
 host="dpe08.fnal.gov" xmlns="http://www-bd.fnal.gov/2011/daqdata">
 <entry lastAccessTime="2011-09-20T18:55:23.232-0500"
  startTime="2011-09-20T18:55:16.228-0500">
  <device canonical="m:outtmp.READING"
   device="m:outtmp" event="U" field="SCALED"
   property="READING" range="[0]"/>
 </entry>
</cache>
```

3.10 Configuration

During the startup procedure the servlet reads initialization parameters from a configuration file located at `<app-dir>/WEB-INF/web.xml`, where `<app-dir>` is path to the application's directory on the DAE server. All of them are listed below. Note however, that usually for the changed parameter to be read an application server must be restarted.

Parameter	Default	Description
cacheCleaningInterval	30000	Time in milliseconds between successive cache cleans.
numberOfDevicesPerJob	128	Maximal number of devices per each DAQ job.
maximalNumberOfJobs	64	Maximal number of DAQ jobs, if it is reached, HTTP 503 Service Not Available error will be thrown.
jobLivingTime	300000	Maximal lifetime of unnecessary (not requested) jobs in milliseconds.
daeUser		DAE username.
daeHost	localhost	DAE hostname.

3.11 Links

1. GetDAQ application page: http://www-bd.fnal.gov/getdaq.
2. Web service URL: http://www-bd.fnal.gov/getdaq/job/.
3. Status page: http://www-bd.fnal.gov/getdaq/status.
4. POST request form: http://www-bd.fnal.gov/getdaq/post.html
5. GET request example: http://www-bd.fnal.gov/getdaq/job/m:outtmp;m:outhum?type=html&iso-time=true&include-drf2=true.
6. Bracket-separated request: http://www-bd.fnal.gov/getdaq/job/(m:outtmp),(m:outhum)?type=html&separator=brackets.