Format efficiency take 2: really clean JSON

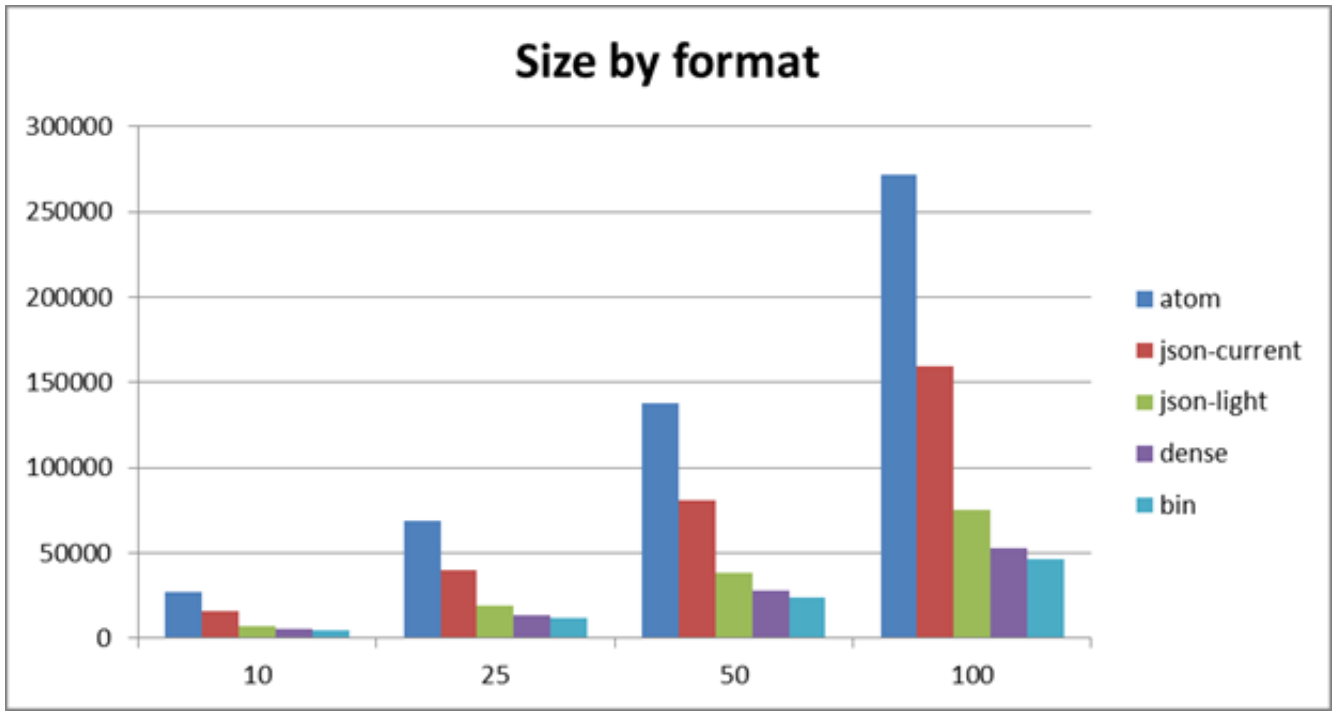
It took me longer than expected to write again about this, but I have another round of measurements and another proposal that goes with it. Since folks want to close on the next version of OData soon, it would be great to iterate quickly on this one so if we all agree we can include it in this version.

Back then we started with some discussion about pros/cons of various options and about what to optimize for (see this thread and this post if you want to see some of the original content). I proposed a JSON-encoded-in-JSON approach that had some fans but also some folks were worried that we might be optimizing for the wrong thing. Based on that I started to look at alternate approaches so I could put more options on the table, and I ended up with something that I think has a lot of potential.

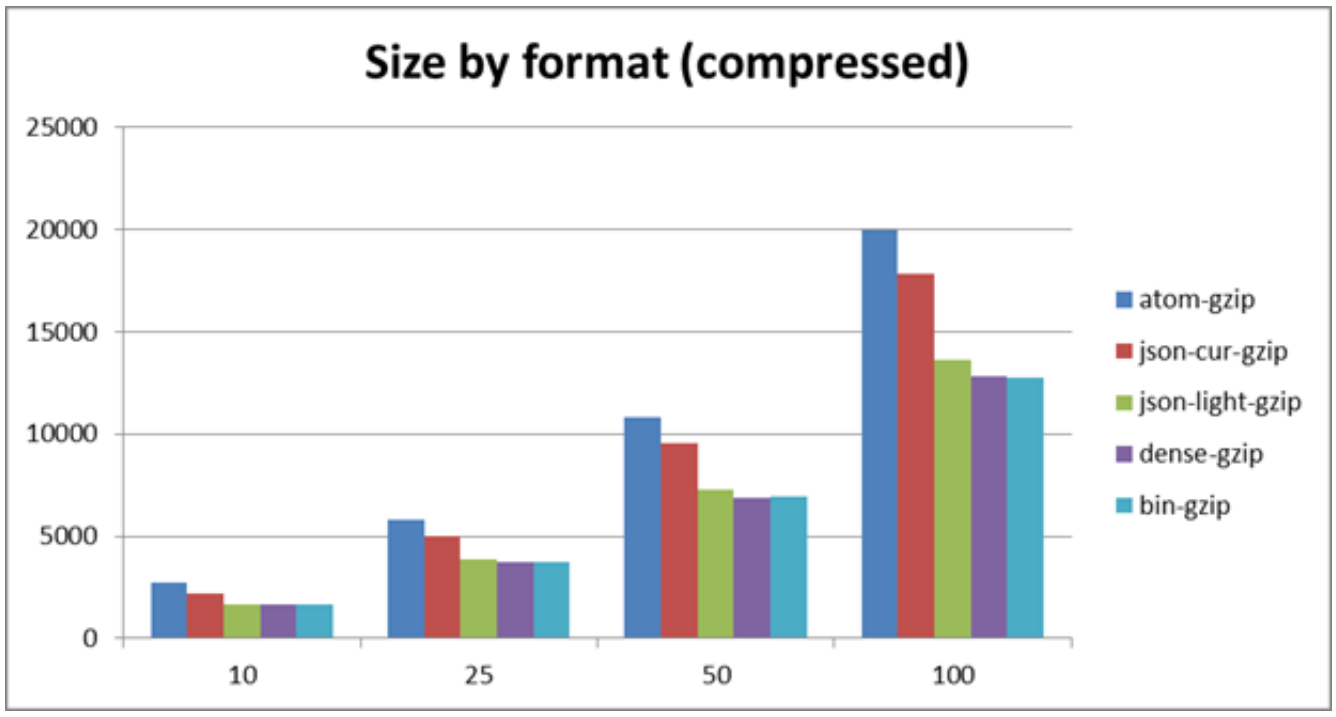
I showed a somewhat half-baked version of this at //BUILD back in September, you can see it by skipping to 00:41:55 in this video.

Trying a different angle

This time I started asking “what if we could serve really clean JSON, just the kind of JSON you'd have in a custom service, but still keep all the richness in semantics of OData?” Control information (particularly URLs) adds lots of bloat to existing JSON format payloads. If you remote it, how do things change? Check out the following chart for a typical OData feed:



So if you remove every bit of control information (e.g. those "__metadata" properties) you end up with a JSON "light" format that's very close to the "dense" format I was exploring before, but without all the weirdness of a custom encoding. In fact, you get very nice and clean JSON, pretty much as if you built a custom endpoint. Now, there is still a small but non-zero difference between "light" and "dense"…what if we combine this with compression? The difference is even smaller:



Now the question is: is it possible to actually describe a format that's clean JSON with no extra stuff in it that still fully maintains OData semantics? I think we can get close enough.

Approach

To put this approach in context I need to establish a key assumption I'm making: there are two big buckets of OData clients, those that just don't care about metadata (because they are too simple to, or because the use out-of-band knowledge and are hardcoded to a particular service) and those that use service metadata in order to maintain decoupling or provide richer functionality.

For the first set, the less stuff we put in our JSON payloads the better, and they've hardcoded knowledge about everything else anyway, so why include it? They can derive URLs from IDs, know when to expect a list versus a single object, etc. Whether hardwiring these assumptions into your application is a good idea depends on the context, I'm not judging here :)

The second set is the interesting one then. The approach for this set of clients can be summarized as follows:

1. All OData clients need to know about two content types, OData metadata and OData data [1]
2. All resources contain a pointer to metadata, so a link to any part of an OData service namespace is fully self-contained and requires no out-of-band knowledge
3. All control information that's uniform enough (most of it) is captured as patterns in metadata
4. Control information that doesn't follow the pattern can be included in any instance, overriding any metadata-described value

This turns this 2-row response (from: http://services.odata.org/OData/OData.svc/Products?$top=2&$inlinecount=allpages&$format=json):


```
{
 "d": {
  "results": [
  {
   "__metadata": {
     "uri": "http://services.odata.org/OData/OData.svc/Products(0)",
     "type": "ODataDemo.Product"
    },
   "ID": 0,
   "Name": "Bread",
   "Description": "Whole grain bread",
   "ReleaseDate": "\/Date(694224000000)\/",
   "DiscontinuedDate": null,
   "Rating": 4,
   "Price": "2.5",
   "Category": {
    "__deferred": {
     "uri": "http://services.odata.org/OData/OData.svc/Products(0)/Category"
    }
   },
   "Supplier": {
```

```
      "__deferred": {
       "uri": "http://services.odata.org/OData/OData.svc/Products(0)/Supplier"
      }
     }
    },
    {
     "__metadata": {
      "uri": "http://services.odata.org/OData/OData.svc/Products(1)",
      "type": "ODataDemo.Product"
     },
     "ID": 1,
     "Name": "Milk",
     "Description": "Low fat milk",
     "ReleaseDate": "\/Date(812505600000)\/",
     "DiscontinuedDate": null,
     "Rating": 3,
     "Price": "3.5",
     "Category": {
      "__deferred": {
       "uri": "http://services.odata.org/OData/OData.svc/Products(1)/Category"
      }
     },
     "Supplier": {
      "__deferred": {
       "uri": "http://services.odata.org/OData/OData.svc/Products(1)/Supplier"
      }
     }
    }],
    "__count": 9
  }
}
```


Into this (note that we also propose we drop the “d” wrapper):


```
{
 "__servicemetadata": "http://services.odata.org/OData/OData.svc/$metadata#ODataDemo.DemoService.Products",
 "results": [
  {
   "ID": 0,
   "Name": "Bread",
   "Description": "Whole grain bread",
   "ReleaseDate": "\/Date(694224000000)\/",
   "DiscontinuedDate": null,
   "Rating": 4,
   "Price": "2.5"
  },
```

```
 {
  "ID": 1,
  "Name": "Milk",
  "Description": "Low fat milk",
  "ReleaseDate": "\/Date(812505600000)\/",
  "DiscontinuedDate": null,
  "Rating": 3,
  "Price": "3.5"
 }],
 "__count": 9
}
```

In the best case all control information goes away. In order to be able to reestablish it, we put one URL per response (in "__servicemetadata") that contains a link to where to find instructions if you want to interpret this document as an OData response with full fidelity. A client can follow the metadata link and using patterns described there reconstruct all URLs, ETags, types, etc. If a given object has something different, e.g. a link that doesn't follow the pattern, or it's an instance of a subtype, then you just add that piece of data (e.g. "__metadata": { "type": "some.subtype" }).

Capturing control information as patterns

I mentioned patterns several times already. Let me make this more concrete. As we discussed before in the OData mailing list, we're adding support for annotations to metadata using vocabularies. In order to support this JSON-based "light" format we introduce a vocabulary that captures how to derive all bits of control information from the regular object data. We'll have the details of every pattern documented in the official spec, but here are a few to show what they look like.

This one shows the base URL for the service, and is used for all relative URLs in other patterns:

```
<ValueAnnotation Term="odata.urls.baseurlexpression" Target="ODataDemo.DemoService">
 <String>http://services.odata.org/OData/OData.svc/</String>
</ValueAnnotation>
```

These two show two URL construction rules, one to obtain the URL of a collection (a set) and one to obtain the URL of an individual element within that collection:

```
<ValueAnnotation Term="odata.urls.setexpression" Target="ODataDemo.DemoService.Products"
            String="Products/"/>

<ValueAnnotation Term="odata.urls.keylookupexpression" Target="ODataDemo.DemoService.Products">
 <Apply Function="KeyConcat">
  <String>(</String>
  <Path>ID</Path>
  <String>)/</String>
 </Apply>
</ValueAnnotation>
```

Finally, here's one that's not a URL but a plain value, in this case the ETag for each element (doesn't apply to the "Product" type, but included here as an example):

```
<ValueAnnotation Term="odata.json.etagexpression" Target="ODataDemo.DemoService.Products">
 <Apply Function="Concat">
  <String>W/"</String>
  <Apply Function="RawValue">
   <Path>Version</Path>
  </Apply>
  <String>"</String>
 </Apply>
</ValueAnnotation>
```

Note that an interesting side-effect of this approach is that removes any knowledge of the server URL namespace from clients. In the past OData clients had to choose between the higher coupling that came from knowing the URL conventions of the server and losing the query capabilities. Now that patterns are captured in metadata a client that knows about both data and metadata content types can derive all URLs from patterns. This removes the coupling and makes it possible for servers to have their own URL conventions as long as they can be represented with annotations (yes, it means you can have a server that uses "/" instead of "(" and ")" if you want, for those that were always unhappy with parenthesis :))

Summary

We discussed a JSON format that's clean and lean and doesn't need a special coding/decoding step and still preserves a lot of the compactness. We achieve this by moving control information that's regular enough to metadata in the form of patterns, and by linking data and metadata so clients need no out-of-band knowledge. The approach also allows servers to have different URL conventions without causing OData clients to lose any functionality.

That do you think? As usual, the OData mailing list is the best place for debate.

-pablo

[1] We had long debates with folks here about introducing "application/odata+json". It seems that this will just cause things to break more for the majority of users, so staying with application/json is probably the right thing. More on this in a future post.

This entry was written by Pablo Castro, posted on Thursday, January 19, 2012 Bookmark the permalink.

Ecosystem
Consumers
Producers
Live Services

Sample Services
Sample Code
Developers
Libraries
Articles & Videos
Validator
Documentation
Protocol
URI Conventions
Atom Format
JSON Format
Community
News
FAQ
Mailing List
Terms of Use | Privacy Statement | Contact Us | Creative Commons Attr.-Noncomm.-No Deriv. Works 3.0