BOSS REST API Spec, rev 3

This is a rendering of the BOSS API spec, proposed last week in a separate document, into a “REST”-style HTTP service. A development version of the service is currently running at the base URL noted below -- please ask Ray Pete for the credentials needed to access this service.

One major change to the way the original BOSS API spec was outlined is the global conversion of URNs in the original spec to URLs in this API. This is a required change for a “REST”-style API, but if the use of URLs doesn’t fulfill a downstream requirement this would be a reasonable criticism to mention within this document.

A Note about URLs

All URLs are relative to the base
`https://boss-dev.broadinstitute.org/`

for the BOSS development system. So, for example, the path
`/objects/12345`

is actually accessed at the path
`https://boss-dev.broadinstitute.org/objects/12345`

Authentication and Access Control

The BOSS web service can only be accessed by an authenticated user, and maintains its own set of users and credentials. The expectation is that these “users” are actually service

accounts (such as “picard” or “vault”), operator accounts (such as “BroadIT” or “genomebridge”) or test accounts on the dev system. They (the BOSS users) do not correspond to end-user accounts. Accounts are configured and managed as part of the system configuration.

Authentication is done via Basic Auth (username/password) to gain access to the service. The username that is authenticated for access to the service is the same identity that is used for access control to an object within BOSS.

Resources

The BOSS API has been made "RESTful" by identifying a single core class of resources, namely "objects". These resources exist on a storage platform, which can be either “opaqueURI”, “localStore” or “cloudStore”. The methods from the BOSS API spec have been translated into HTTP methods on these resources.

Objects

An object resource represents data in the storage platform, either an uninterpreted URI or an object in an objectstore. Objects can be created (see Creating Objects below), retrieved/described (Describing Objects), and resolved into URLs (Resolving Objects). Operations on an object resource are performed by HTTP methods on the URL of the resource, which contains the Object ID assigned by BOSS.

Object names are not guaranteed to be unique, but are queryable via the API. When an Object is created (registered) in BOSS, it is assigned a unique ID (unique across all objects within the BOSS implementation), which is used to reference the object in later calls to the API. The Object ID will be a string that can be embedded into an url without encoding (e.g., a UUID). The exact implementation of the Object ID is internal to BOSS and the caller should treat them as an atomic string.

Representation

An Object consists of the following fields:

- `"objectId": <string>` - a string containing the ID assigned by BOSS when the object was created (i.e. registered in BOSS). This ID is guaranteed to be unique across all objects in BOSS. It will be a string that is suitable for embedding into an url without encoding (e.g. a UUID). The exact implementation of the Object ID is internal to BOSS and the caller should treat it as an atomic string. The Object ID will be no more than 1024 characters long.

  The current BOSS implementation uses UUIDs for the object identifiers and formats them in the standard way as a 36 character string, consisting of 5 hex numbers, as in “xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx”. To be compatible with future

implementations, the caller should not be dependent on this format.

- `"objectName": <string>` - a string assigned by the caller that is a display name for this object. This name is not guaranteed (or required) to be unique. The name can be used to query the API.

- `"storagePlatform": <string>` - specifies the storage platform containing the underlying data object that BOSS is referencing. Currently, it must be one of
  - `"localStore"`
  - `"cloudStore"`
  - `"opaqueURI"`

- `"directoryPath": <string>` - a string that should be a valid URI. You could, for example pass a file:///path/to/some/file URI that references the file containing the underlying data object, if you set storagePlatform to “opaqueURI”. This field is only present for “opaqueURI” objects, and it is an error to specify it when registering a “localStore” or “cloudStore” object. Note that the caller is responsible for interpreting the URI in the appropriate context. BOSS does not do any checking of the validity of the path that is given, and merely stores it and returns it on resolve.

- `"sizeEstimateBytes": <integer>` - a number representing an estimate of the size of the underlying data object. Currently this is stored and returned but not processed by BOSS. Future versions may use this as a hint to allocate storage.

- `"ownerId": <string>` - the name of the user who is the “owner” of the data. Currently this is stored and returned, but not validated or interpreted by BOSS.

- `"readers": [ <string> ]` - an array of 1 or more strings which are BOSS usernames. This field lists the (BOSS) users who have read access to this BOSS object.

- `"writers": [ <string> ]` - an array of 1 or more strings which are BOSS usernames. This field lists the (BOSS) users who have write access to this BOSS object.

Methods

Describing Objects

GET `/objects/:objectId`

Required Headers

Header Name	Header Value
Accept	A media type spec that includes "application/json"

Response:
The caller should expect a 200 OK response, with a JSON representation of the Object resource. If no object is found with the specified :objectId, the caller will get a 404 response. The caller must have read access (i.e. the username of the caller must be in the “readers” list) to the referenced object, or a 403 response is returned. If the object has been deleted, the response will be 410.

For example,

```
GET /objects/768cf9ca-7ff6-4c23-9100-4e66ceb45d81
```

might return the following JSON representation of a Object resource.

```
{
  "objectId": "768cf9ca-7ff6-4c23-9100-4e66ceb45d81",
  "objectName": "Test Object #1",
  "storagePlatform": "localStore",
  "sizeEstimateBytes": 500,
  "ownerId": "tdanford",
  "readers": [
    "picard",
    "vault"
  ],
  "writers": [
    "picard"
  ]
}
```

Creating Objects

Objects are created by POSTing a JSON document to the /objects URL:

```
POST /objects
```

Required Headers:

Header Name	Header Value

Content-Type	application/json
Accept	a media type spec which includes "application/json"

Message Body:

A JSON representation of an Object, *except* for the "objectId" field (which is generated). For example, to create the Object whose representation was retrieved above, we POSTed the following JSON file

```
{
  "ownerId" : "tdanford",
  "objectName" : "Test Object #1",
  "storagePlatform" : "localStore",
  "sizeEstimateBytes" : 500,
  "readers" : [ "picard", "vault" ],
  "writers" : [ "picard" ]
}
```

to the URL

```
https://boss-dev.broadinstitute.org/objects
```

Response:
The user should expect a 201 OK response, with the JSON representation of the object, which will now have the objectId field instantiated.

To create an opaqueURI object, the “storagePlatform” field must be set to “opaqueURI”, and an additional JSON field is required to specify the URI, such as

```
  "directoryPath" :
“file:///seq/picard_aggregation/C1124/ABCDE-123-N/v2/C1124-123-N.bam”,
```

If the `"directoryPath"` attribute is specified when “storagePlatform” is not “opaqueURI”, BOSS will return a 400 Bad Request error response.

Deleting Objects

Objects are deleted by executing the HTTP DELETE method against their URL. For an object in the “localStore” or “cloudStore” platform, it will also delete the underlying object. For an “opaqueURI” object, deleting whatever the URI references is the caller’s responsibility.

`DELETE` `/objects/:objectId`

Required Headers: None
Response: The user should expect a 200 OK response. The message body echoes the objectId.

The caller must have write access (i.e. the username of the caller must be listed in the “writers” field of the object) in order to delete it, or a 403 response will be returned. If no object with the given :objectId is found, a 404 response will be returned. If BOSS is unable to delete the underlying object from the object storage (e.g. non-transient network failure, or other error from the objectstore server), it will return an appropriate 50x error, and the entry for this object will not be deleted from BOSS.

Resolving Objects

Objects are "resolved" into a URL which can be used to either upload or download the resource. This behaves differently based on storage platform. For an “opaqueURI” object, it will simply return the URI that was passed when the object was created. For a “localStore” or “cloudStore” object, resolve will return a pre-signed URL for either uploading or downloading the object, as indicated by the operation. If no operation argument is specified, the default is “download”.

POST /objects/:objectId/resolve

Required Headers:

Header Name	Header Value
Accept	A media type specification that includes "application/json"

Message Body:

A “resolution request”, which is a JSON object containing the following fields:
* validityPeriodSeconds
* httpMethod
* contentType (optional)
* contentMD5Hex (optional)

For example, the following object is an example of a valid resolution request:

```
{
  "validityPeriodSeconds": 1000,
  "httpMethod": "GET"
}
```

or,

```
{
  "validityPeriodSeconds": 1000,
  "httpMethod": "PUT",
  "contentMD5Hex": "00112233445566778899aabbccddeeff"
}

or,

{
  "validityPeriodSeconds": 1000,
  "httpMethod": "PUT",
  "contentType": "text/plain"
}
```

Valid values for “httpMethod” are one of { GET, PUT, HEAD }, and the “validityPeriodSeconds” value must be strictly positive. If specified, the “contentType” value must be a valid mime type, and must be passed in the header to resulting request to objectUrl. If specified, “contentMD5Hex” value must be a valid 32 character hexadecimal string.

NOTE: The specific header to pass for the MD5 in the resulting HTTP request to objectUrl is specific to the object store. For example S3 compatible object stores require the MD5 to be base 64 encoded, not hexadecimal. For example “Content-MD5: ABEiM0RVZneImaq7zN3u/w==”.

Response:
The client should expect a 200 OK and a response similar to the following:

```
{
  "objectUrl": "http://boss-ci.genomebridge.org/objects/obj123",
  "validityPeriodSeconds": 1000
}
```

(NOTE: The objectUrl example above is not an actual example of a pre-signed URL -- they’re much uglier.)

Where `"objectUrl"` is a pre-signed url generated to access the object for the desired operation. To generate a GET or HEAD url, the caller must have read access to the object. To generate a PUT url, the caller must have write access to the object. If the caller does not have the correct access, a 403 response will be returned. If the object with the specified :objectId is not found, a 404 will be returned (or a 410, if the object has been deleted).

For ”opaqueURI” objects, the resolve function will return the URI passed when the object was created without interpretation.

Copying Objects

The Copy service is similar to the Resolve service: It returns an URL that can be used to write data to a BOSS Object. In this case, however, the ultimate source of the data is some other object in the same object store. Currently this works only for cloudStore objects stored in GCS.

POST /objects/:objectId/copy

Required Headers:

Header Name	Header Value
Accept	A media type specification that includes "application/json"

Message Body:

A “copy request”, which is a JSON object containing the following fields:
* validityPeriodSeconds
* locationToCopy

For example, the following object is an example of a valid copy request:

```
{
  "validityPeriodSeconds": 1000,
  "locationToCopy": "/myBucket/myKey"
}
```

Response:
The client should expect a 200 OK and a response similar to the following:

```
{
  "uri": "https://some/really/long/complicated/url?with=lotsOfParams"
}
```

You should then do a PUT to that URL. You must include a header with the location to copy:

```
x-goog-copy-source: /myBucket/myKey
```

The signature of the pre-signed URL anticipates that you will provide this header, and the call will fail if you fail to provide it. Sorry you have to repeat yourself. The PUT should have an empty body (which is unusual for a PUT, but that’s the spec).

After a (potentially long) period of time, you’ll get an OK returned, and your BOSS object will be populated with a copy of the data from the other object.

Updating Objects

Object resources can be updated by POSTing a modified representation of the resource to the URL. Only the ownerId, readers, and writers fields can be modified through POSTs to an Object resource.

```
POST /objects/:objectId
```

Required Headers:

Header Name	Header Value
Content-Type	application/json
Accept	any media type specification that includes "application/json"

Message Body:
A JSON object containing the values of the fields to *update* (as noted above, this can only be the ownerId, readers, and writers fields). Any fields *not* specified in the body will *not* be changed. Illegal fields in the post will generate a 400 Bad Request response. The request may also return 403 (no write permission), 404 (no such object), or 410 (object has been deleted).

Querying for Objects by Name

Objects for which a user has read access can be queried by GETting that name from the /objects URL with the query parameter name=objectName. This returns a 200 OK response, and a JSON list of object descriptions as documented in the Describing Objects section. If you have no readable objects stored by that name, the query produces a 404 Not Found response.

```
GET /objects?name=objectName
```

Required Headers:

Header Name	Header Value
Accept	any media type specification that includes "application/json"