BOSS REST API Spec, rev 3

A Note about URLs
Resources
Objects
    Representation
    Methods
        Describing Objects
        Creating Objects
        Deleting Objects
        Resolving Objects
        Updating Objects

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 “filesystem” or “objectstore”. 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 a file in the file system or an object in the objectstore. Objects can be created (the "registerObject" and “registerFsObject API methods), retrieved/described (the "describeObject" API method), and resolved into URLs (the "resolve" method). 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, and are not 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, and cannot 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
  - `"objectstore"`
  - `"filesystem"`

- `"directoryPath": <string> -` a string that should be a valid unix filepath that references the file containing the underlying data object, if the storagePlatform was set to “filesystem”. This field is only present for “filesystem” objects, and it is an error to specify it when registering an “objectstore” object. Note that the caller is responsible for interpreting the filepath 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 with “file:” prepended 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.

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": "objectstore",
  "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" : "objectstore",
  "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 a file system object, the “storagePlatform” field must be set to “filesystem”, and an additional JSON field is required to specify the file path such as

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

If the `"directoryPath"` attribute is specified when “storagePlatform” is not “filesystem”, 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 “objectstore” platform, it will also delete the underlying object. For a “filesystem” object, deleting the file is the callers responsibility.

`DELETE` `/objects/:objectId`

Required Headers: None
Response: The user should expect a 200 OK response, with an empty message body.

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 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 a filesystem object, it will generate a "file:" URL that points to a path on the shared filesystem, and it is the caller's responsibility to ensure that the URL is only used in a context where the file path is valid. For an objectstore 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

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

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

Valid values for "httpMethod" are one of { GET, PUT, HEAD }, and the "validityPeriodSeconds" value must be strictly positive.

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 example pre-signed URL)

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.

For file system objects (i.e. storagePlatform="filesystem"), the resolve function will return a URL of the form

```
  "objectUrl": "file:<directoryPath>",
```
where <directoryPath> is the string specified when the object was registered. The path is not validated or formatted in any way, other than pre-pending "file:".

Updating Objects

Object resources can be updated by POSTing a modified representation of the resource to the URL. Currently, in order to match the "reassignOwnership" and the "setPermissions" methods of the API spec, *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.