Error codes

Documentation
04/28/2014

API Error Responses ¶

A successful API call will result in one of several possible HTTP success codes such as:

- `200 OK` – Operation successful and done
- `201 Created` – New document created
- `202 Accepted` – Job/task accepted and will be completed sometime in the future (i.e. is not necessarily complete now)
- `301 Moved Permanently` – Document's name has successfully been changed (renamed and thus "moved")

There are also a number of possible failure responses. Which one it is tells you something about exactly what the problem is.

- And if `apiCaller.parseRespBody()` is successful, there is even more information in the `"status"` section of the response.

Example of `400 Bad Request`

- Perhaps you did not construct the values for the 3 authorization parameters correctly?
- `Bad Request` is one of the more common errors when there is a problem with how you are making the request.
  - Sometimes even things like invalid payloads or payload syntaxes (e.g. XML) may tell you `Bad Request` and can be solved by following documentation & examples). These errors should really be one of the more specific types like `Not Acceptable` or `Unsupported Media Type`

```
1 # Deliberately cause the wrong gbToken value to be computed:
2 apiCaller = BRL::Genboree::REST::ApiCaller.new(gbHost, "", dbrcRec[:user], dbrcRec[:password] + "corrupt")
3
4 # Say we want to GET all the docs in a GenboreeKB
5 apiCaller.setRsrcPath("/REST/v1/grp/{grp}/kb/{kb}/coll/{coll}/docs")
6   #=> #<Net::HTTPBadRequest 400 Bad Request readbody=true>
```

- Message in response payload is pretty clear and fully of suggestions

```
1 {
2   "data": {
3   },
4   "status": {
5     "msg": "BAD_TOKEN: The gbToken provided is not correct for: this URL, the timestamp, the indicated user, and user
6     "statusCode": "Bad Request"
7   }
8 }
```

Example of `403 Forbidden` – Not a Member of Group

- If you aren't a member of the Group the GenboreeKB is in, then you can't GET, PUT, etc, resource that are within
  - Unless it has been made public or unlocked [and you have the `gbKey`), in which case you can do read-only operations like GET

```
1 # This user is not [currently] in the "ARJ.a" group:
2 apiCaller = BRL::Genboree::REST::ApiCaller.new(gbHost, "", "andrewj_guest", guestPassword)
3
4 # Try a GET
5 apiCaller.setRsrcPath("/REST/v1/grp/{grp}/kb/{kb}/coll/{coll}/docs")
6    #=> #<Net::HTTPForbidden 403 Forbidden readbody=true>
```

- The `"status"` section message is pretty clear about what's wrong:

```
1 {
2   "data": {
3   },
4   "status": {
5     "msg": "FORBIDDEN: The Genboree database \"clgbr - arj test 1\" is private and you are not a member of the group
6     "statusCode": "Forbidden"
7   }
8 }
```

Example of `403 Forbidden` – Member, but only "subscriber" (read-only)

- Even if you are a member, you may not have a role that permits certain operations. e.g. `subscribers` can't write/delete resources.

```
# We have made "andrewj_guest"@ a subscriber in the group
apiCaller = BRL::Genboree::REST::ApiCaller.new(gbHost, "", "andrewj_guest", guestPassword)

# Going to add a new doc
apiCaller.setRsrcPath("/REST/v1/grp/{grp}/kb/{kb}/coll/{coll}/doc/{docId}")
dataFile = JSON.parse( File.read( "./tmp.clingen/clgbr_file/clgbr999.json" ) )

# Try a PUT of a new doc
apiCaller.put(
  { :grp => "ARJ.a", :kb => "clgbr - arj test 1", :coll => "clgbr.a (2014-04)", :docId => "999|MSH6:NM_000179:c.G161
  dataFile.to_json
)
   #=> #<Net::HTTPForbidden 403 Forbidden readbody=true>
```

- Again, if you do a `apiCaller.parseRespBody()` and look in the `"msg"` field of the status section, you will see an informative message

```
{
  "data": {
  },
  "status": {
    "msg": "FORBIDDEN: The username provided does not have sufficient access or permissions to operate on the resourc
    "statusCode": "Forbidden"
  }
}
```

Example of `404 Not Found`

- When a resource doesn't exist (i.e. the URL path doesn't indicate a real, existing resource) you can get a `Not Found` response.
- E.g. There is no such group, no such GenboreeKB, or even no such document with the provided ID.
- Sometimes this is because of bad/non-robust approaches to escaping of users' names for things when making the URL.
  - Failing to escape a value being placed in a URL path.
  - Escaping too many times.
- Or being careless with case-sensitivity.

```
# Let's go back to our non-guest user
apiCaller = BRL::Genboree::REST::ApiCaller.new(gbHost, "", dbrcRec[:user], dbrcRec[:password])

# Say we want to GET all the docs in a GenboreeKB
# * BUT we are so smart, we are going to build the URL ourselves (non-robust--cause of many Genboree code bugs by sl
#    who don't use ApiCaller in more robust ways)
# escape the kb name
kb = CGI.escape(gbKbName)
# build URL
apiCaller.setRsrcPath( "/REST/v1/grp/" + CGI.escape("ARJ.a") + "/kb/" + CGI.escape(kb) + "/coll/" + CGI.escape("clgb

# Call GET to retrieve the docs
apiCaller.get( { :grp => "ARJ.a", :kb => "clgbr - arj test 1", :coll => "clgbr.a (2014-04)" } )
   #=> #<Net::HTTPNotFound 404 Not Found readbody=true>

# Wow, we are so smart.
```

- As usual we can parse the response body and get some feedback from the `"status"` section.
- Maybe it gives some clue as to our bug.

```
{
  "data": {
  },
  "status": {
    "msg": "NO_KB: There is no user knowledgebase \"clgbr%20-%20arj%20test%201\" in user group \"ARJ.a\" (or perhaps
    "statusCode": "Not Found"
  }
}
```

(hmm, what are those weird %20 in our kb name...I thought we had a nice kb name...)

Example of `405 Method Not Allowed`

- HTTP defines 6 methods for operating on resources: `GET`, `PUT`, `DELETE`, `HEAD`, `OPTIONS`, and ~~`POST`~~ (ill-defined, poor for REST based APIs; never use)
- But not all methods can be used on all resources.
  - Especially currently, when `PUT` and `DELETE` are exposed only for documents and not for collections, models, GenboreeKbs, etc.
- If we call an unsupported method for a resource, the failure should be clear.

```
# Say we will try to delete all the docs in the GenboreeKB (ouch!)
```

```
apiCaller.rsrcPath #=> "/REST/v1/grp/ARJ.a/kb/clgbr%2520-%2520arj%2520test%25201/coll/clgbr.a%20%282014-04%29/docs?"

# Do the delete, destroying everyone's hard work in a blink:
apiCaller.delete( { :grp => "ARJ.a", :kb => "clgbr - arj test 1", :coll => "clgbr.a (2014-04)" } )
   #=> #<Net::HTTPMethodNotAllowed 405 Method Not Allowed readbody=true>

# phew!
```

- And the `"status"` section has sensible explanation:

```
{
  "data": {
  },
  "status": {
    "msg": "METHOD_NOT_ALLOWED: The 'DELETE' operation is not implemented for the \"http://10.15.5.109/REST/v1/grp/AR
    "statusCode": "Method Not Allowed"
  }
}
```

Example of `406 Not Acceptable`

- A less common error occurs when the document in the request payload (of say a PUT) is invalid and thus unacceptable.

```
# We are going to try to put a new doc
apiCaller.setRsrcPath("/REST/v1/grp/{grp}/kb/{kb}/coll/{coll}/doc/{docId}")
dataFile = JSON.parse( File.read( "./tmp.clingen/clgbr_file/clgbr999.json" ) )

# Say that: the document has:
# - an incorrectly spelled root identifier property ("documentId" instead of "documentID"...case sensitive)
# - regardless, has an empty value for that root property (also not allowed)
dataFile['documentId'] = dataFile['documentID']
dataFile.delete('documentID')
dataFile['documentID']['value'] = ''

# Now try to PUT this document, using the id we THINK is in the actual document (but isn't)
apiCaller.put(
  { :grp => "ARJ.a", :kb => "clgbr - arj test 1", :coll => "clgbr.a (2014-04)", :docId => "999|MSH6:NM_000179:c.G161
  dataFile.to_json
)
   #=> #<Net::HTTPNotAcceptable 406 Not Acceptable readbody=true>

```

- As usual, the `"status"` section of our API envelope explains what is wrong. both in the general and in the specific

```
{
  "data": {
  },
  "status": {
    "msg": "BAD_DOC: the GenboreeKB doc you provided in the payload does not match the document model for the \"clgbr
    "statusCode": "Not Acceptable"
  }
}
```

Example of `415 Unsupported Media Type`

- Another rare error is return when your payload is something that *does not* follow our API standard and is thus un-parseable and not understood.
  - E.g. You used some unsupported syntax like XML (with its 10x larger size)
  - E.g. You didn't make use of our *Standard API Envelope* and your data object is not provided under the `"data"` key in the payload
  - E.g. You didn't send valid JSON or send multibyte/Unicode characters or something

```
# Again, we will try to put our new document
apiCaller.rsrcPath #=> "/REST/v1/grp/{grp}/kb/{kb}/coll/{coll}/doc/{docId}?"
dataFile = JSON.parse( File.read( "./tmp.clingen/clgbr_file/clgbr999.json" ) )

# And we build its JSON ourselves. Maybe by hand, maybe with buggy code. Regardless, say the JSON is incorrect (synt
docJson = dataFile.to_json
docJson.chomp!('}')  # - oops, missing a }
docJson << "\n#\n#"  # wrong kind of comments for JSON

# Do the PUT, using our own JSON string
# Now try to PUT this document, using the id we THINK is in the actual document (but isn't)
apiCaller.put(
  { :grp => "ARJ.a", :kb => "clgbr - arj test 1", :coll => "clgbr.a (2014-04)", :docId => "999|MSH6:NM_000179:c.G161
  docJson
```

```
15 )
16    #=> #<Net::HTTPUnsupportedMediaType 415 Unsupported Media Type readbody=true>
```

- Because it's a JSON syntax error, the `"status"` feeback has details from the JSON parser class.
- But it's pretty clear in the first sentence what has gone wrong.

```
1 {
2   "data": {
3   },
4   "status": {
5     "msg": "There was an error parsing the request body, please check the format and entity type. (JSON::ParserError:
6     "statusCode": "Unsupported Media Type"
7   }
8 }
```

NOTE: if you see `500 Internal Server Error` then this indicates a bug in the server-side code. There may be a problem with the request or document, but it should have been noticed & handled [with a `4xx` error] but wasn't. Something bad happened.

- Still, try to parse the response and examine the `"status"` message. There may be useful info about the bug there!

Files

New file