V2 API

This document describes how to use version 2 of the Bright API.

The Bright API provides programmatic access to a variety of functions provided by the Bright platform, most significantly read and write access to the Bright database. Via the API it is possible to register users, assign courses, generate launch URLs, and query results.

This document details how to get connected to the Bright Server, how to authenticate, and what functions are available via which API calls. It also covers what to do when things go wrong, and how to get more information.

More information about Bright and the Bright platform can be accessed from the Aura Homepage.

General Usage

The Bright API is used to interact with Bright Server. Typical things you might do with Bright are

- access data about learners, courses, registrations.
- You might be querying data or pushing data in.

Regardless, all Bright API calls function in the same way. Fundamentally, the Bright API is a RESTful API, so much of what is covered here should seem familiar if you've worked with a RESTful API before. If not, you may want to familiarize yourself with the concepts here: http://en.wikipedia.org/wiki/Representational*state*transfer .

> **Representational State Transfer** (**REST**) is a style of software architecture for distributed systems such as the World Wide Web. REST has emerged as a predominant Web service design model.
>
> The term *representational state transfer* was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation.[1][2] Fielding is one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification versions 1.0 and 1.1.[3][4]
>
> Conforming to the REST constraints is generally referred to as being "RESTful".[5]

Source: Wikipedia

RESTful API

The Restful api of bright is built around our main resource types:

- courses
- registrations
- users
- api keys
- invitations

for each resource type, we follow the following model, where a specific path, http verb and action are consistent depending on the bright resource you are accessing.

HTTP Verb	Path	action	used for
GET	/{{resource}}	index	display a list of all {{resource}} [Note: Bright does not implement this canonical REST action].
GET	/{{resource}}/new	new	return an HTML form for creating a new {{resource}}
POST	/{{resource}}	create	create a new {{resource}}
GET	/{{resource}}/gcreate	create	create a new {{resource}} with a GET request. This is an alias to the POST get request provided to allow the {{resource} to be created from a web browser without using a POST. A variety of cross-domain scripting issues can be addressed using the GET create alias (gcreate).
GET	/{{resource}}/:id	show	display a specific {{resource}}
GET	/{{resource}}/:id/edit	edit	return an HTML form for editing a {{resource}}
PUT	/{{resource}}/:id	update	update a specific {{resource}}
GET	/{{resource}}/gupdate/:id	update	update a specific {{resource}} via a GET request. See the comment on gcreate above.
DELETE	/{{resource}}/:id	destroy	delete a specific {{resource}}

List of Resouces Available

- Bright Api Keys (/bright/api/v2/api_key)

- Course Provider (/bright/api/v2/course_provider)
- Courses (/bright/api/v2/course)
- Registrations (/bright/api/v2/registation)
- Realm User (/bright/api/v2/realm_user)
- Embed Templates (/bright/api/v2/embed_template) [Note: in v3, this will be renamed simply "template"].
- Invitations (/bright/api/v2/invitation)

Special Considerations for Cross-Browser Scripting Restrictions

One thing about the RESTFul architecture is does not interoperate well with cross browser scripting restrictions in modern browsers. Specifically, you are probably aware that you can not POST or PUT data via JSONP which means you can interact with a *pure* RESTFul API from a web page delivered from a web server other the one hosting the Bright API.

To alleviate this problem, we provide two "GET"-oriented aliases, gcreate and gupdate, that work identically to create and update, and are available via HTTP GET calls.

Quick Start

In the section we will quickly work throw the steps of connecting to the Bright API, authenticating, and generating API calls.

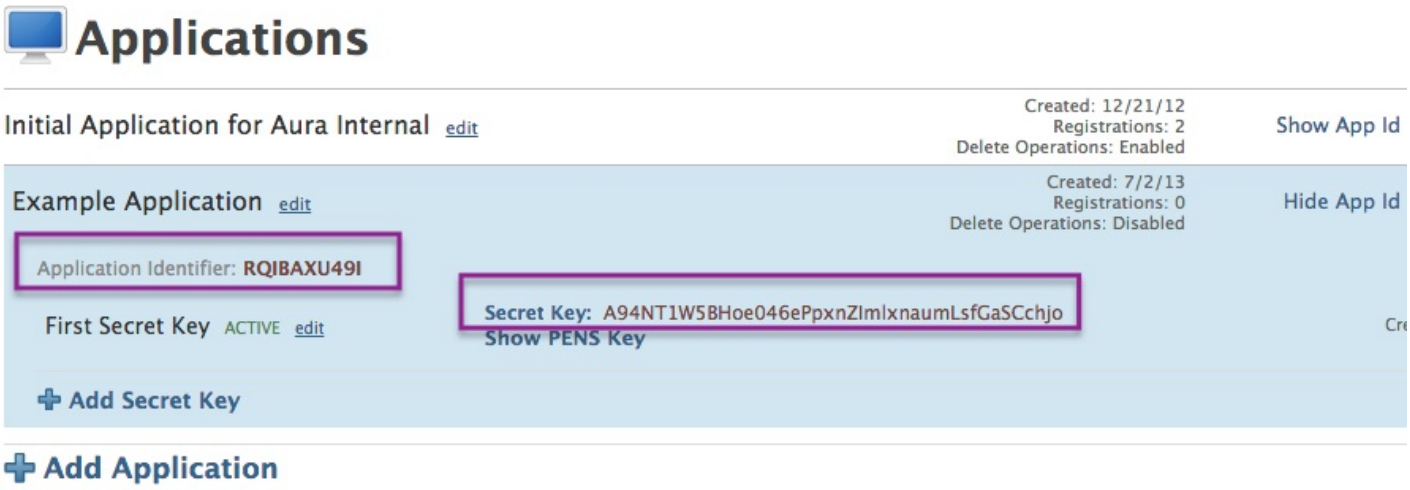
Prerequisite Data

- Your Bright API URL. This will be provided to you and is based on where your Bright server is installed, and what DNS entry is used to access it. Don't know it? Ask us at support@aura-software.com!

If you are using a SCORMCloud course provider, API access can be provided using your SCORMCloud APP ID and secret key. These are available from Aura Support, or from the SCORMCloud console available at https://cloud.scorm.com.

- The SCORMCloud APP ID for your SCORMCloud Application.
- The SCORMCloud Secret Key for your SCORMCloud Application.

Here's an example of fetching these values from the SCORMCloud administration console:



You can access this by selection "Apps" from the SCORMCloud administration console left hand menu.

You can also access Bright with a Bright realm app id and keysee section.

Certain functionality can only be accessed with a realm key, such as invitations or the utilization of multiple course providers.

For the purposes of this example, we will use the following:

- Bright API URL: http://[BRIGHT URL]/bright/api/v2
- SCORMCloud APPID: **RQIBAXU49I**
- SCORMCloud Secret Key: **nCwrTDSy1MzaeyhN0TFfi3uH3huzlu6CNmyHUG5N**

With the above information, we can already use our API. To do so, we can test this straight from curl a command line tool that is easy to install on most systems.

When learning the Bright API, we recommend you start by assembling some simple curl commands from your command line in order to get a feel for what is possible.

So let's get our list of course from our API:

```
curl 'http://[BRIGHT URL]/bright/api/v2/course.xml?sc_app_id=RQIBAXU49I&
sc_secret_key=nCwrTDSy1MzaeyhN0TFfi3uH3huzlu6CNmyHUG5N'
```

If you've executed this correctly, you'll get a result like:

And the result:

```
<?xml version="1.0" encoding="UTF-8"?>
<scorm-cloud-courses type="array">
  <scorm-cloud-course>
    <course-provider-id type="integer">6</course-provider-id>
```

```
      <created-at type="datetime">2012-11-26T12:10:40Z</created-at>
      <custom></custom>
      <id type="integer">184</id>
      <metadata>{"title":"ENT Foundation - Post Training QUIZ",
                 "description":"",
                              "duration":"0",
                              "typicaltime":"0",
                              "keywords":null}
          </metadata>
      <registration-count type="integer">1</registration-count>
      <sc-course-id>1-507727747154e</sc-course-id>
      <size type="integer">157758</size>
      <title>System Test Course</title>
      <updated-at type="datetime">2013-01-17T16:20:10Z</updated-at>
      <versions type="integer">-1</versions>
    </scorm-cloud-course>
  </scorm-cloud-courses>
```

Specifying a results format.

Fetching an XML result

To fetch your results in XML format, append a '.xml' to the url, *before* the request parameters. For example

```
http://[BRIGHT URL]/bright/api/v2/course.xml
```

Fetching a JSON result

Let's say if you are using the API from Javascript, and you'd like your results back as JSON. Easy, just rewrite the URL to use 'course.json' instead of 'course.xml'

```
curl 'http://[BRIGHT URL]/bright/api/v2/course.json?
sc_app_id=RQIBAXU49I&
sc_secret_key=nCwrTDSy1MzaeyhN0TFfi3uH3huzlu6CNmyHUG5N'
```

And the result (formatted for readibility:

```
[
 {
         "created_at":"2012-11-26T12:10:44Z",
         "id":187,
         "registrations":2,
         "sc_course_id":"1-5098f07394cbd",
         "course_provider_id":6,
         "size":1599415,
         "title":"Proktologie",
         "updated_at":"2012-11-27T22:30:40Z",
         "versions":-1
 },
 {
         "created_at":"2012-11-29T16:11:53Z",
         "id":200,
         "registrations":1,
         "sc_course_id":"1-50b63e903dd43",
         "course_provider_id":6,
         "size":2211931,
         "title":"Proktologie",
         "updated_at":"2012-12-07T15:48:43Z",
         "versions":-1
 }
]
```

Access Modes

The Bright API can be accessed to two different ways:

- SCORMCloud App ID and Secret Key
- Bright API Key
- [Brigh Realm GUID and Secret Key) (#access-modes/via-realm-guid-and-secret-key)

If you do not specify either the api key/secret key pair **OR** the Bright API key, you will receive HTTP code 401 (Unauthorized) in your response.

If you specify multiple authentication models, you will received a 501 (not implemented).

If you specify an unknown SCORMCloud APPId and Secret key, you will receive a 500 (server error).

Via SCORMCloud App ID and Secret Key

In all of the examples so far, we've been using direct access to the API using the SCORMCloud secret key and app id. Generally speaking, this is fine for server side code that is secured and publicly accessible. You should NEVER share the APP ID and secret key since this give complete access to all of your data.

Example:

```
curl 'http://[BRIGHT URL]/bright/api/v2/course.json?sc_app_id=\
RQIBAXU49I&sc_secret_key=nCwrTDSy1MzaeyhN0TFfi3uH3huzlu6CNmyHUG5N'
```

The above example will show all courses for the course provider defined by the SCORMCloud data (app id, secret key).

Via Bright API Key

If you want to access the Bright API from some browser side Javascript, using the SCORMCloud or Bright Realm secret key is no good. NEVER put your Secret Key into the browser via Javascript.

Instead the Bright API allows you to create an authentication token that you can use to send to the browser. When you generate this token, typically you specify

- the SCORMCloud app id/secret key OR bright realm secret key
- optionally the user

These application tokens are disabled after a short period of time. Do not hard code the use of a Bright API key, as these keys expire after a period of time.

Here's an example of getting an API token. This first call creates a special access token to a specific user for a specific SCORMCloud application. The key that is returned is suitable for embedding in a web page for use by browser side Javascript.

```
curl 'http://[BRIGHT URL]/bright/api/v2/api_key/create?sc_app_id=RQIBAXU49I
&sc_secret_key=nCwrTDSy1MzaeyhN0TFfi3uH3huzlu6CNmyHUG5N
&user_email=admin@aura-software.com'
(returns) df8d1350a6b31378a86b967767f4bba1
```

You can now omit the secret key and app id from subsequent calls and just use the API key:

```
curl 'http://[BRIGHT URL]/bright/api/v2/course.xml?\
api_token=bdb273e9cdace9698c34d97070cb392d'
```

This API token is now "bound" to the access level specified when the key was created.

Via Realm ID and Secret Key

Your Bright realm ID and secret key will be furnished to you by Bright support. For many functions it is not necessary. If you aren't working with invitations, creating user records on the fly, or using multiple course providers, you probably won't need a Realm key. Note you do not need to create a user to assign a user to a course. This is done on the fly and if you aren't populating custom metadata for a user, it is not necessary to pre-populate the user.

If you need to use a realm ID and secret key, it will be provided to you by Aura support. You can file a request at support@aura-software.com

For the purposes of this example, we will use the following:

- realm_guid: sJLtP8Zt8G0Sbz9kxPjQ
- realm*secret*key: PcVQflTCUIbe3ps2T86KXAzvXzdpFcgs5Mvku03uZ8w

With these, lets create an api key 'bound' to this authentication level:

```
curl 'http://localhost:3000/bright/api/v2/api_key/gcreate?
realm_guid=sJLtP8Zt8G0Sbz9kxPjQ&
realm_secret_key=PcVQflTCUIbe3ps2T86KXAzvXzdpFcgs5Mvku03uZ8w'
```

And the response:

```
{
 "access_token":"2a84a6ddb229c13bc945874b69fab8ba",
 "course_provider_id":null,
 "created_at":"2013-07-03T00:25:37Z",
 "expires_at":null,
 "id":2038,
 "realm_id":4,
 "token_type":null,
 "updated_at":"2013-07-03T00:25:37Z",
 "user_id":null
}
```

We can now use the "access_token" returned as an API key in an insecure situation:

```
curl 'http://localhost:3000/bright/api/v2/course?
```

```
api_key=2a84a6ddb229c13bc945874b69fab8ba'
```

Or in a secure situation, you can use the realm key directly:

```
curl 'http://localhost:3000/bright/api/v2/course?
realm_guid=sJLtP8Zt8G0Sbz9kxPjQ&
realm_secret_key=PcVQflTCUIbe3ps2T86KXAzvXzdpFcgs5Mvku03uZ8w'
```

API Modules

Note not all RESTFul function are implemented.

Registration

Method: index

The index method allows the api user to query registrations.

HTTP Model:

Verb	Form
GET	(http|https)://BRIGHT_URL/bright/api/v2/registration[.format]?param1=value1&...

Parameters

Parameter	Example	Notes
__access method__	api_key=[an api key created previously	see [Access Modes](#access-modes)
last_only	last_only=t	when included in the parameter string, we will return a single registration record, that being the last one created. Useful in cases where multiple registrations are found and only the most recent one should be returned.
sc_course_id	sc_course_id=a_scorm_cloud_course_id	filter registrations to the specified course id.
learner_id	learner_id=jane.doe@me.com	filter registrations to the specified learner_id.
refresh_if_launched	refresh_if_launched=t	if the last launch date of this registration is newer than the last crawl date, this will force the record to recrawl prior to returning a result.
crawl	crawl=t	forces the registration to recrawl

Example

```
curl -w "%{http_code}" 'http://localhost:3000/bright/api/v2/registration.json?
api_key=2a84a6ddb229c13bc945874b69fab8ba&
learner_id=bret@aura-software.com&
sc_course_id=16-4fbd9ea698bce
```

HTTP Codes

Code	Description
200	Success; items returned
401	If you do not specify a valid api_key, sc_app_id/sc_secret_key or realm_guid/realm_secret_key, you will receive HTTP error 401 (unauthorized).
500	An illegal request, such as a malformed argument.

Method: create

Creates a new scorm cloud registration. Can be used in a single call that can be used to return an existing registration if one exists, or create a new one if it does not.

Parameters

Parameter	Example	Notes
__access method__	api_key=[an api key created previously]	see Access Modes
	sc_app_id=XXXXYYYYZZZ [a	

sc_app_id	sc_app_id=XXXXYYYYZZZ [a valid app ID]	SCORMCloud application ID (required)
sc_secret_key	sc_secret_key=XXXXYYYYZZZ [a valid app secret key]	SCORMCloud application secret key (required)
sc_course_id	sc_course_id=16-4fbd9ea698bce	SCORMCloud course ID (required)
learner_id	learner_id=bret@aura-software.com	email address of user (required)
dont_duplicate	dont_duplicate=t	check for an existing registration for this course and user, just return that if found. (optional)
check_scorm_cloud	check_scorm_cloud=t	check the SCORMCloud API for existing registrations as well as the local Bright database (optional). This can be useful in test scenarios where Bright SCORMCloud crawler is not functioning, and it is desirable not to create a lot of duplicate registrations. Typically not used in production.
fname	fname=bret	Can be used to set the first name in the SCORMCloud registration.
lname	lname=weinraub	Can be used to set the last name in the SCORMCloud registration.

HTTP Codes

Code	Description
201	Success; item created
401	If you do not specify a valid sc_app_id/sc_secret_key or realm_guid/realm_secret_key, you will receive HTTP error 401 (unauthorized).
403	In some cases an additional error message is returned (see below).
404	You provided a sc_course_id that can't be found for this access method, or none at all. If you are using the create (and not gcreate alias), make sure you are posting the data correctly.
500	If you specify a user_email that is not valid, or a valid sc_course_id, you will receive a 500 server error.

In case of errors accessing the SCORMCloud API, a message including the error is returned (error code 403):

```
{
"error_code":"4",
"error_message":"The maximum number of registrations has been reached.",
"error_url":"http://cloud.scorm.com/api?email=..."
}
```

Please note if you are using JSONP, since there are limited facillities to capture errors via JSONP, we will return error code 200, with the error block above. Otherwise you might find it difficult to correctly handle this error.

Example

For a successful request returned will be an XML or JSON document of the SCORMCloud registration. Note this example is using the gcreate alias (using an HTTP get).

```
curl -w "%{http_code}"
'http://localhost:3000/bright/api/v2/registration/gcreate.json?
api_key=2a84a6ddb229c13bc945874b69fab8ba&
learner_id=bret@aura-software.com&
sc_course_id=16-4fbd9ea698bce&
sc_app_id=XXXYYYYZZZ&
sc_secret_key=nCwrTDSy1MzaeyhN0TFfi3uH3huzlu6CNmyHUG5N&
dont_duplicate=t'
{
 "attempts"=>2,
 "complete"=>"complete",
 "course_result"=>'<?xml version="1.0" encoding=...?> <rsp>...</rsp>',
 "crawl_error_msg"=>nil,
 "crawl_status"=>"success",
 "crawl_type"=>"course",
 "created_at"=>"2012-12-21T08:39:12Z",
 "full_result"=>nil,
 "id"=>5336,
 "last_crawled_at"=>"2012-12-12T16:33:33Z",
 "learner_id"=>"bret@aura-software.com",
 "number_of_crawl_errors"=>0,
 "sc_completed_at"=>nil,
 "sc_course_id"=>"20-NSFoundationPostTrainingQuiz",
 "sc_created_at"=>"2012-12-12T16:32:04Z",
```

```
 "sc_deleted"=>nil,
 "sc_err_code"=>nil,
 "sc_error_message"=>nil,
 "sc_last_accessed_at"=>nil,
 "sc_registration_id"=>"79c0dd35-139-048258-16e-9a1561b0a85d",
 "score"=>99.2,
 "course_provider_id"=>10,
 "course_id"=>509,
 "success"=>"succeeded",
 "totaltime"=>100.0,
 "updated_at"=>"2012-12-21T08:39:12Z"
}
```

Course

Method: Index

The index method allows the api user to fetch a course list.

HTTP Model:

Verb	Form
GET	(http|https)://BRIGHT_URL/bright/api/v2/course[.format]?param1=value1&...

Parameters

Parameter	Example	Notes
__access method__	api_key=[an api key created previously]	see [Access Modes](#access-modes)

Example

```
curl -w "%{http_code}" 'http://localhost:3000/bright/api/v2/course.json?
learner_id=bret@aura-software.com&
sc_course_id=16-4fbd9ea698bce
```

HTTP Codes

Code	Description
200	Success; items returned
401	If you do not specify a valid api_key, sc_app_id/sc_secret_key or realm_guid/realm_secret_key, you will receive HTTP error 401 (unauthorized).
500	An illegal request, such as a malformed argument.

Realm User

For learners, its generally not necessary to prepopulate realm users as creating a registration implicitly creates the user.

It is possible to create them via the API and also manipulate their custom field as well.

Method: create

Creates a new realm user. You must use a realm guid and secret key (or api key creating bound to a realm key) to create a realm user.

Parameters

Parameter	Example	Notes
__access method__	api_key=[an api key created previously]	see Access Modes. You cannot use a SCORMCloud app-id/secret key or user bound api key to create a realm user.
email	email=bret@aura-software.com	email address of the user (required)
realm_role_id	realm_role_id=1	Define the realm role. Not required, defaults to '1' (learner).
custom	custom='{this:that}'	Define a custom field for this user. Custom fields are freeform metadata available to be defined for the user.

HTTP Codes

Code	Description
201	Success; item created
302	Item already exists, not modified.
500	If you specify a user_email that is not valid, or a valid sc_course_id, you will receive a 500 server error.

Example

```
curl -w "%{http_code}"
'http://localhost:3000/bright/api/v2/realm_user/gcreate.xml?
api_key=2a84a6ddb229c13bc945874b69fab8ba&
email=bretx@aura-software.com&
custom=foo'
<?xml version="1.0" encoding="UTF-8"?>
<realm-user>
  <created-at type="datetime">2013-07-03T05:26:21Z</created-at>
  <custom>foo</custom>
  <id type="integer">1743</id>
  <realm-id type="integer">4</realm-id>
  <realm-role-id type="integer">1</realm-role-id>
  <updated-at type="datetime">2013-07-03T05:26:21Z</updated-at>
  <user-id type="integer">1686</user-id>
</realm-user>
201
```

Method: update (gupdate)

Update a realm user. You must use a realm guid and secret key (or api key creating bound to a realm key) to update a realm user.

Parameters

Parameter	Example	Notes
__access method__	api_key=[an api key created previously]	see Access Modes. You cannot use a SCORMCloud app-id/secret key or user bound api key to create a realm user.
email	email=bret@aura-software.com	email address of the user (required)
custom	custom='{this:that}'	Define a custom field for this user. Custom fields are freeform metadata available to be defined for the user.

HTTP Codes

Code	Description
200	Success; item created
404	Realm user cannot be found..
500	If you specify a user_email that is not valid, or supply unexpected parameters, you will receive a 500 server error.

Example

```
curl -w "%{http_code}"
'http://localhost:3000/bright/api/v2/realm_user/gupdate.xml?
api_key=2a84a6ddb229c13bc945874b69fab8ba&
email=bretx@aura-software.com&
custom=foox'
200
```

API Key

The API Key interface allows for the creation of API keys for Bright Server.

Method: create

HTTP Model:

Verb	Form
Post	(http|https)://BRIGHT_URL/bright/api/v2/api_key?param1=value1&...

Parameters

Parameter	Example	Notes
sc_app_id		(optional) You must use either sc_app_id and sc_secret_key OR realm_guid and realm_secret_key.
sc_secret_key		(optional)
realm_guid		(optional) You must use either sc_app_id and sc_secret_key OR realm_guid and realm_secret_key.
realm_secret_key		(optional)
		(optional). When specifying a user email when

user_email	user_email=jane.doe@me.com	generating an API key, this api key will be "bound" to this user and will be unable to access data unrelated from the user. If you do not set user_email, the generated token will have unlimited (administrative) access to the SCORMCloud application or Bright Realm specified. If you specify an email that is new to the system, an account is automatically created for that user, and the new user is attached as a subscriber class user to the Bright Realm attached to the this SCORM Cloud application.

Return Data

Returns a JSON document of the new record. HTTP Code is set to 201 (item created).

Example:

```
curl -w "%{http_code}" -d 'sc_app_id=RQIBAXU49I
&sc_secret_key=nCwrTDSy1MzaeyhN0TFfi3uH3huzlu6CNmyHUG5N
&user_email=admin@aura-software.com' http://[BRIGHT URL]/bright/api/v2/api_key/

{
 "access_token":"a440b1ad868bc76716d22b4b827db77e",
 "created_at":"2012-12-18T12:16:29Z",
 "expires_at":null,
 "id":139,
 "course_provider_id":6,
 "token_type":null,
 "updated_at":"2012-12-18T12:16:29Z",
 "user_id":12
}
201
```

Note that this is a POST request, so you must configure your client appropriately. In the case of curl, this is accomplished with the **-d** flag.

The '-w %{http_code}' piece allows you to view the returned HTTP code.

HTTP Codes

Code	Description
201	Success; item created
401	If you do not specify a valid sc_app_id/sc_secret_key or realm_guid/realm_secret_key, you will receive HTTP error 401 (unauthorized).
500	If you specify a user_email that is not valid, you will receive a 500 server error.

Debugging Errors

The Bright API, if it's not happy with its input, isn't going to give you that much information as to why. It doesn't produce error messages other than HTTP codes. This is primarily for security reasons. Since we listen on public ports, we don't want people probing the api, and using the error messages as a guide to on how to gain access.

From curl, if you put an invalid response, you'll get no data back:

```
curl 'http://[BRIGHT URL]/bright/api/v2/course.json?sc_app_id=RQIBAXU49I
&sc_secret_key=nCwrTDSy1MzaeyhN0TFfi3uH3huzlu6CNmyHUG5N'
```

We asked for a jsonc format, which of course doesn't exist.

The only information you will get is an HTTP error code, in this case 406, 'Not Acceptable'.

Fetching HTTP Error Codes from a GET Request

You can see the HTTP response code for a GET request in curl with the -I flag:

```
curl 'http://[BRIGHT URL]/bright/api/v2/course.json?
sc_app_id=RQIBAXU49I&
sc_secret_key=nCwrTDSy1MzaeyhN0TFfi3uH3huzlu6CNmyHUG5N' -I
 HTTP/1.1 406 Not Acceptable
 Content-Type: text/html; charset=utf-8
 X-Ua-Compatible: IE=Edge,chrome=1
 Cache-Control: no-cache
 X-Request-Id: d4da4e1010b84640e1657b731ada79a3
 X-Runtime: 0.004287
 Date: Fri, 07 Dec 2012 23:04:10 GMT
```

```
  X-Rack-Cache: miss
  Content-Length: 0
  Server: WEBrick/1.3.1 (Ruby/1.9.3/2012-04-20)
  Connection: Keep-Alive
```

You will also get no data from a request that matches no data. But in this case, the HTTP code will be 200

```
curl 'http://[BRIGHT URL]/bright/api/v2/course.json?sc_app_id=RQIBAXU49I\
&sc_secret_key=nCwrTDSy1MzaeyhN0TFfi3uH3huzlu6CNmyHUG5N&title=nosuchcourse'
```

[]

```
curl 'http://[BRIGHT URL]/bright/api/v2/course.json?sc_app_id=RQIBAXU49I
&sc_secret_key=nCwrTDSy1MzaeyhN0TFfi3uH3huzlu6CNmyHUG5N&title=nosuchcourse' -I
  HTTP/1.1 200 OK
  Content-Type: application/json; charset=utf-8
  X-Ua-Compatible: IE=Edge,chrome=1
  Etag: "d751713988987e9331980363e24189ce"
  Cache-Control: max-age=0, private, must-revalidate
  X-Request-Id: 73a4966a2a62ca4e70316f6a68645b51
  X-Runtime: 0.006220
  Date: Fri, 07 Dec 2012 23:06:40 GMT
  X-Rack-Cache: miss
  Content-Length: 0
  Server: WEBrick/1.3.1 (Ruby/1.9.3/2012-04-20)
  Connection: Keep-Alive
```

Fetching HTTP Error Codes from a POST Request

Since we use a RESTful API, some operations require a POST HTTP verb. For whatever reason, the -I flag in curl will not show you the HTTP return code for a POST.

Instead use the following:

```
curl -w "%{http_code}"  -d 'sc_app_id=RQIBAXU49Ix&sc_secret_key=\
nCwrTDSy1MzaeyhN0TFfi3uH3huzlu6CNmyHUG5N&user_email=newuser@aura-software.com' \
http://[BRIGHT URL]/bright/api/v2/api_key
```

-w %{http_code} is the key part of that.

Debugging Authentication Errors

If you receive a 401 code from the API, the server itself has recorded the authentication error in the server log. If you are completely stumped, this is the place to look.

An error like this should exists:

```
Processing by ScormCloudCourseController#index as JSON
  Parameters: {"api_key"=>"bogus!", "sc_course_id"=>"course1"}
   [1m [36mApiKey Load (0.4ms) [0m   [1mSELECT "api_keys".* FROM
  "api_keys" WHERE "api_keys"."access_token" = 'bogus!' LIMIT 1 [0m
Unauthorized: api token not found
Filter chain halted as :restrict_access rendered or redirected
Completed 401 Unauthorized in 2ms (ActiveRecord: 0.4ms)
```

Definitely give us a shout if you can't get an API call to work at support@aura-software.com.