cURL Demo

Mark Bykerk Kauffman

cURL Demos

Introduction

The rest demo script demonstrates authenticating a REST application, management and use of the authorization token, and creating, updating, discovering, and deleting the Data Source and User Learn objects. A video presentation using this material plus an explanation of how to register your REST application on the Developer Portal, and how to configure your application's REST integration on your Learn development system, is available here. Here is the slide deck

Prerequisites

- You must register a developer account and application in the Developer Portal
- You must register your application in Learn
- You must also configure the script as outlined in the README for the project

This cURL command-line demonstration shows you how to:

- Authenticate
- Create a Data Source
- Create, Read, and Update a User
- Delete created objects in reverse order of create - user, datasource.
- Create a grade column and add a grade.
- Upload a file to a content area.

All generated output is sent to the terminal.

This is not meant to be a cURL tutorial. It will not teach you to use cURL. It will, however, give a Developer familiar with cURL the knowledge necessary to make a complete set of CRUD operations to the Learn REST endpoints.

Assumptions

This help topic assumes the Developer:

- is familiar with cURL.
- has a REST-enabled Learn instance.

Walkthrough

To build an integration with the Learn REST Web Services, regardless of the programming language of choice, can really be summed up in two steps:

1. Use the Application Key and Secret to obtain an **OAuth 2.0 access token**, as described in the Basic Authentication document.

OR

1. Use the authorizationcode endpoint to log into Learn with a given user's credentials, then obtain an **OAuth 2.0 access token**, using the application key/secret, that only allows access to the system based on that particular user's entitlements as described in the Three-Legged OAuth document.
2. Call the appropriate REST endpoint with the **OAuth 2.0 access token** and necessary data to perform a given action.

Every example is shown in bold face, the JSON result is shown in italics.

Authorization and Authentication

OAuth2 Basic (Two Legged)

The REST Services rely on OAuth 2.0 Bearer Tokens for authentication. A request is made to the token endpoint with a Basic Authorization header containing the base64-encoded key:secret string as its key. The token service returns a JSON object containing the Access Token, the Token Type, and the number of seconds until the token expires. The token is set to expire after one hour, and subsequent calls to retrieve the token will return the same token with an updated expiry time until such time that the token has expired. There is no refresh token and currently no revoke token method.

You can do this with cURL with the following command line. We use the -k parameter to ignore issues caused by the self-signed certificate.

```
curl -k --user <key>:<secret> --data "grant_type=client_credentials"
https://<LearnHost>/learn/api/public/v1/oauth2/token
```

Example:

```
curl -k --user d128e50d-c91e-47d3-a97e-9d0c8a87fb5d:kLpiuq34320jqreaiJIRoareASELERREv56
--data "grant_type=client_credentials"
 ↪  https://localhost%3A9877/learn/api/public/v1/oauth2/token
```

```
{
  "access_token": "ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz",
  "token_type": "bearer",
  "expires_in": 3444
}
```

The JSON response is serialized into the Token object as shown above, and you may then retrieve those values from that object by using copy and paste. You will use the value given for the access_token when calling the services as shown in the **Calling Services** section.

OAuth2 (Three Legged) - 3LO

To work with 3LO you will need both a browser, to log into Learn and retrieve a code, and a terminal window for your cURL commands. Note: In step 1 the client_id is NOT the Application ID. It IS the Application Key.

Step 1. GET a code using your browser - CAUTION: You MUST set the scope parameter.

Put the GET request in your browser's address field. Craft a URL in the following format:

```
https://<LearnHost
  >//learn/api/public/v1/oauth2/authorizationcode?redirect_uri=<REST APP URI>
     & response_type=code & client_id=<your **app key**> & scope=read</your></REST></LearnHost>
```

Example:

Place the URI below in your browser address field:

```
https://bd-partner-a-original.blackboard.com/learn/api/public/v1/oauth2/authorizationcode?
redirect_uri=https://localhost & response_type=code &
 ↪  client_id=d128e50d-c91e-47d3-a97e-9d0c8a77fb5d & scope=read
```

At this point you are requested to log in.

As of 2018.04.19 there is a bug with the cookie-acceptance pop up that will require you to accept the pop-up and do the above a second time. Once your browser has the cookie-acceptance cookie set for a given Learn system the work flow will carry on as follows.

For this example, after you log in your browser will be directed to:

`https://localhost/?code=XYTdmQcSGrggzujJm2Ccf8C7dKyqKc7Q`

(The code will be different every time.) If you provided some other host name in the redirect_uri, then that host name would be shown instead of localhost. Now you have the code that you will use to retrieve the access token for all additional REST calls.

NOTE regarding scope: The developer portal shows an example scope parameter as "read write offline". Using curl this would be `& scope=read_write-offline`

We have to URL encode the space character.

Step 2. POST to get the OAuth2 Access Token

You will use your application's key and secret, and the access code from step

1. Notice that the grant_type is authorization_code in this case. Also note that the redirect_uri value MUST match the value provided in Step 1. For our Example we will use `https://localhost`.

```
curl -k --user <key>:<secret> --data "grant_type=authorization_code"
https://<LearnHost>/learn/api/public/v1/oauth2/token
```

Example:

```
curl -k --user d128e50d-c91e-47d3-a97e-9d0c8a77fb5d:kLpiuq34320jqreaiJIRoareASELERREv56
--data "grant_type=authorization_code"
https://bd-partner-a-original-new.blackboard.com/learn/api/public/v1/oauth2/token?
 ↪  code=ItmqfxQiA9dzIDNwNoNYseM5GNRHHlfa & redirect_uri=https://localhost
```

```
{"access_token":"Cdf83I0dwRoweXuY1dZDbbW0f0WmDmuF", "token_type":"bearer", "expires_in":3599,
 ↪  "scope":"read", "user_id":"7ac650e5bd0d467882943ed06fbfe72c"}
```

The JSON response is serialized into the Token object as shown above, and you may then retrieve those values from that object by using copy and paste. You will use the value given for the access_token when calling the services as shown in the **Calling Services** section.

Calling Services

The individual service calls are handled in succession here. We'll use copy and paste to create the JSON object in the form of a string, use cURL for generating the appropriate HTTP Request, and shipping it to Learn. We'll see the JSON response come back to stdout on the console.

End points are generally defined as `/learn/api/public/v1/<object type>/<objectId>`. Object ID can be either the pk1, like `_1_1`, or as the batchuid. This value should be prepended by externalId:, like `externalId:test101`.

For example, to retrieve a course by the pk1 `_1_1`, you would call **GET /learn/api/public/v1/courses/_1_1**. To retrieve by the batchuid `test101`, you would call **GET /learn/api/public/v1/courses/externalId:test101.**

Create is sent to Learn as a HTTP POST message with a JSON body that defines the object. The endpoint should omit the objectId, as this will be generated on creation.

Read is sent to Learn as a HTTP GET message with an empty body. The endpoint should include the objectId being retrieved.

Update is sent to Learn as a HTTP PATCH message with a JSON body that defines the object. The endpoint should include the objectId being updated.

Delete is sent to Learn as a HTTP DELETE message with empty body. The endpoint should include the objectId being deleted.

Every example is shown in bold face, the JSON result is shown in italics.

DATASOURCES

Create

```
curl -k -X POST -H "Authorization: Bearer <Access Token>" -H "Content-Type: application/json"
--data '{"externalId":"<String>", "description":"<String>"}'
https://<LearnHost>/learn/api/public/v1/dataSources
```

Bearer will be the access_token value we got from the oauth2/token call above.

Example:

```
curl -k -X POST -H "Authorization: Bearer ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz" -H
"Content-Type: application/json" --data '{"externalId":"CURLDSK", "description":"cURL Demo
 ↪  DSK"}'
https://localhost:9877/learn/api/public/v1/dataSources
```

```
{ "id": "_5_1", "externalId": "CURLDSK", "description": "cURL Demo DSK" }
```

Read

```
curl -k -X GET -H "Authorization: Bearer <Access Token>"
https://<LearnHost>/learn/api/public/v1/dataSources/externalId:<externalId from create>
```

Example:

```
curl -k -X GET -H "Authorization: Bearer ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz"
https://localhost:9877/learn/api/public/v1/dataSources/externalId:CURLDSK
```

```
{ "id": "_7_1", "externalId": "CURLDSK", "description": "cURL Demo DSK" }
```

Update

```
curl -k --request PATCH -H "Authorization: Bearer <Access Token>" -H
"Content-Type: application/json" --data '{"externalId":"<String>", "description":"<String>"}'
https://<LearnHost>/learn/api/public/v1/dataSources/[<primary id>|externalId:<String>]
```

Examples:

```
curl -k --request PATCH -H "Authorization: Bearer ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz" -H
"Content-Type: application/json" --data '{"externalId":"CURLDSK", "description":"cURL
 ↪  DEMONSTRATION DSK"}'
 https://localhost:9877/learn/api/public/v1/dataSources/_7_1
```

```
{
  "id": "_7_1",
  "externalId": "CURLDSK",
  "description": "cURL DEMONSTRATION DSK"
}
```

```
curl -k --request PATCH -H "Authorization: Bearer ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz" -H
"Content-Type: application/json" --data '{"externalId":"CURLDSK", "description":"cURL REST DEMO
 ↪  DSK"}'
 https://localhost:9877/learn/api/public/v1/dataSources/externalId:CURLDSK
```

```
{ "id": "_7_1", "externalId": "CURLDSK", "description": "cURL REST DEMO DSK" }
```

Delete (If you are going to run the following examples, do so now before deleting your demo DSK.)

```
curl -k -X DELETE -H "Authorization: Bearer <Access Token>"
https://<LearnHost>/learn/api/public/v1/dataSources/[<primary id>|externalId:<String>]
```

Examples:

```
curl -k -X DELETE -H "Authorization: Bearer ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz"
https://localhost:9877/learn/api/public/v1/dataSources/externalId:CURLDSK
```

With delete, you don't see anything back on the command line to indicate your delete was successful. You can run it again and you'll see that the DSK is truly gone.

```
curl -k -X DELETE -H "Authorization: Bearer ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz"
https://localhost:9877/learn/api/public/v1/dataSources/externalId:CURLDSK
```

```
{"status":404, "message":"Could not find object with ID:
externalId:CURLDSK", "extraInfo":"416d7b944a58482aaed2df50f301861b"}
```

USERS

Create

```
curl -k -X POST -H "Authorization: Bearer <Authorization Token>" -H
"Content-Type: application/json" --data '<JSON to create a User>'
https://localhost:9877/learn/api/public/v1/users
```

Remember, you can always find the JSON specification for these calls at Explore APIs

Example:

```
curl -k -X POST -H "Authorization: Bearer ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz" -H
"Content-Type: application/json" --data '{"externalId":"restdemouser", "dataSourceId":"_7_1"
, "userName":"restdemouser", "password":"xyzzy", "availability":{"available":"Yes"},
"name":{"given":"demo", "family":"user", "title":"Mr"}, "contact":
{"email":"no.one@ereh.won"}}' https://localhost:9877/learn/api/public/v1/users
```

```
{
  "id": "_7_1",
  "uuid": "de2198a86c6645cbafaab13e79529e05",
  "externalId": "restdemouser",
  "dataSourceId": "_7_1",
  "userName": "restdemouser",
  "educationLevel": "Unknown",
  "gender": "Unknown",
  "created": "2016-05-11T21:39:11.518Z",
  "systemRoleIds": ["User"],
  "availability": { "available": "Yes" },
  "name": { "given": "demo", "family": "user", "title": "Mr" },
  "job": {},
  "contact": { "email": "no.one@ereh.won" },
  "address": {},
  "locale": {}
}
```

By sheer coincidence this user's ID, '_7_1', came out to be the same as the DSK ID, '_7_1'.

Note the availability of query parameters in the documentation at https://developer.anthology.com.

Examples:

```
curl -k -X GET -H "Authorization: Bearer bsdcojzT9i4qTaNE0T8ugEBcXE2U8jtu"
https://bd-partner-a-ultra.blackboard.com/learn/api/public/v1/users?dataSourceId=_2_1
 & fields=externalId,userName,studentId
```

```
{"results":[{"externalId":"[abcd001@examity.com](mailto:abcd001@examity.com)",
"userName":"[abcd001@examity.com](mailto:abcd001@examity.com)", "studentId":"ABCD001"},
```

```
{"externalId":"[abcd002@examity.com](mailto:abcd002@examity.com)",
 ↪  "userName":"[abcd002@examity.com]
(mailto:abcd002@examity.com)",
 ↪  "studentId":"[ABCD002@examity.com](mailto:ABCD002@examity.com)"},...
```

On a Windows system, use ^ to escape the & .

```
curl -k -X GET -H "Authorization: Bearer bsdcojzT9i4qTaNE0T8ugEBcXE2U8jtu"
https://bd-partner-a-ultra.blackboard.com/learn/api/public/v1/users?
dataSourceId=_220_1^ & fields=externalId,userName,studentId
```

Read

```
curl -k -X GET -H "Authorization: Bearer <Authorization Token>"
https://localhost:9877/learn/api/public/v1/users/[<primary id>|externalId:<String>]
```

Examples:

```
curl -k -X GET -H "Authorization: Bearer ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz"
https://localhost:9877/learn/api/public/v1/users/externalId:restdemouser
```


```
{
  "id": "_7_1",
  "uuid": "de2198a86c6645cbafaab13e79529e05",
  "externalId": "restdemouser",
  "dataSourceId": "_7_1",
  "userName": "restdemouser",
  "educationLevel": "Unknown",
  "gender": "Unknown",
  "created": "2016-05-11T21:39:11.518Z",
  "systemRoleIds": ["User"],
  "availability": { "available": "Yes" },
  "name": { "given": "demo", "family": "user", "title": "Mr" },
  "job": {},
  "contact": { "email": "no.one@ereh.won" },
  "address": {},
  "locale": {}
}
```


```
curl -k -X GET -H "Authorization: Bearer ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz"
https://localhost:9877/learn/api/public/v1/users/_7_1
```


```
{
  "id": "_7_1",
  "uuid": "de2198a86c6645cbafaab13e79529e05",
  "externalId": "restdemouser",
  "dataSourceId": "_7_1",
  "userName": "restdemouser",
  "educationLevel": "Unknown",
  "gender": "Unknown",
  "created": "2016-05-11T21:39:11.518Z",
  "systemRoleIds": ["User"],
  "availability": { "available": "Yes" },
  "name": { "given": "demo", "family": "user", "title": "Mr" },
  "job": {},
  "contact": { "email": "no.one@ereh.won" },
  "address": {},
  "locale": {}
}
```

Update

```
curl -k --request PATCH -H "Authorization: Bearer <Access Token>" -H "Content-Type:
 ↪  application/json"
--data '<JSON Data for a User>' https://<LearnHost>/learn/api/public/v1/users/
[<primary id>|externalId:<String>]
```

Example:

```
curl -k --request PATCH -H "Authorization: Bearer ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz" -H
"Content-Type: application/json" --data '{"externalId":"restdemouser", "dataSourceId":"_7_1",
"userName":"restdemouser", "availability":{"available":"Yes"}, "name":{"given":"Jane",
"family":"Demo", "title":"Ms"}, "contact":{"email":"no.one@ereh.won"}}'
 https://localhost:9877/learn/api/public/v1/users/_7_1
```


```
{
  "id": "_7_1",
  "uuid": "de2198a86c6645cbafaab13e79529e05",
  "externalId": "restdemouser",
  "dataSourceId": "_7_1",
  "userName": "restdemouser",
  "educationLevel": "Unknown",
  "gender": "Unknown",
  "created": "2016-05-11T21:39:11.518Z",
  "systemRoleIds": ["User"],
  "availability": { "available": "Yes" },
  "name": { "given": "Jane", "family": "Demo", "title": "Ms" },
  "job": {},
  "contact": { "email": "no.one@ereh.won" },
  "address": {},
  "locale": {}
}
```


Delete

```
curl -k -X DELETE -H "Authorization: Bearer <Access Token>"
https://<LearnHost>/learn/api/public/v1/users/[<primary id>|externalId:<String>]
```

Example:

```
curl -k -X DELETE -H "Authorization: Bearer ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz"
https://localhost:9877/learn/api/public/v1/users/externalId:restdemouser
```

As with the DSK delete there is no output to the console when the delete is successful. But, you can see that the user is gone by running the delete again:

```
curl -k -X DELETE -H "Authorization: Bearer ti3EVMVQO4RqdAgcpmODZdvjvHuuBHDz"
https://localhost:9877/learn/api/public/v1/users/externalId:restdemouser
```


```
_{"status":404, "message":"Could not find object with ID:
externalId:restdemouser", "extraInfo":"e709f6982af744dfae18d42f68e73fb1"}_
```


COURSE GRADES

(Only in SaaS)

Create Grade Column

```
curl -k -X POST -H "Authorization: Bearer <Authorization Token>" -H "Content-Type:
 ↪  application/json" --data '<JSON to create a Grade Column>'
 ↪  [https://saashost.blackboard.com/learn/api/public/v1/courses/{courseId}/gradebook/columns
```

Example:

```
curl -k -X POST -H "Authorization: Bearer uFMWyuwgItXL0UzMo6AHG0zOhm0yvfys" -H "Content-Type:
 ↪  application/json" --data '{"id": "7", "externalId": "co7extId", "name": "co7name",
 ↪  "description": "co7desc", "externalGrade": true, "created":"2016-06-30T05:34:56.497Z",
 ↪  "score": {"possible": 77, "decimalPlaces":0}, "availability": {"available": "Yes"},
 ↪  "grading": {"type":"Manual", "due": "2016-07-01T05:34:56.498Z", "attemptsAllowed":0,
 ↪  "scoringModel": "Last", "anonymousGrading": { "type": "None",
 ↪  "releaseAfter":"2016-07-11T05:34:56.498Z"}}}'
 ↪  https://partner-smoke-test-a.blackboard.com/learn/api/public/v1/courses/_50_1/gradebook/columns
```


```
{"id":"**_129_1**", "externalId":"co7extId", "name":"co7name", "description":"co7
desc", "externalGrade":true, "created":"2016-07-01T05:45:37.730Z", "score":{"poss
ible":77.0, "decimalPlaces":0}, "availability":{"available":"Yes"}, "grading":{"t
ype":"Manual", "due":"2016-07-01T05:34:56.498Z"}}
```


Add/Update Column Grade

(PATCH will create a grade where there isn't one.)

```
curl -k --request PATCH -H "Authorization: Bearer <Access Token>" -H "Content-Type:
 ↪  application/json" --data '<JSON Data for a Column Grade>' https://<Learn
 ↪  Host>/learn/api/public/v1/courses/{courseId}/gradebook/columns/{columnId}/users/{userId}
```

Example:

```
curl -k --request PATCH -H "Authorization: Bearer uFMWyuwgItXL0UzMo6AHG0zOhm0yvfys" -H
 ↪  "Content-Type: application/json" --data '{"text": "A", "score": 77, "notes": "Great",
 ↪  "feedback": "GoodWork!", "exempt":true}'
 ↪  https://partner-smoke-test-a.blackboard.com/learn/api/public/v1/courses/_50_1/gradebook/
columns/_129_1/users/_66_1
```


```
{
  "userId": "_66_1",
  "columnId": "_129_1",
  "status": "Graded",
  "score": 77.0,
  "notes": "Great",
  "feedback": "GoodWork!",
  "exempt": true
}
```


ACCESS CONTENT

- See Access Content Attachments

COURSE COPY

```
curl -k -X POST -H "Authorization: Bearer <Authorization Token>" -H "Content-Type:
 ↪  application/json" --data '<JSON for the course to copy into>'
 ↪  https://saashost.blackboard.com/learn/api/public/v1/courses/{courseId}/copy
```

Example:

```
curl -k -X POST -H "Authorization: Bearer RcHoSkh8EH9UKo6iCHWXDpCKrmCyOIwr" -H "Content-Type:
 ↪  application/json" --data '{"courseId":"mbk-test-copied201910111502"}'
 ↪  https://partner-smoke-test-a.blackboard.com/learn/api/public/v1/courses/courseId:mbk-test/copy
```

A new course with courseId: mbk-test-copied201910111502 is created. The contents of mbk-test are copied into it.

NOTE: You can NOT copy contents into an already existing course. For example we used the GUI to create a course with a courseId: mbk-test- copied201910111508. Then we attempted to run the following:

```
curl -k -X POST -H "Authorization: Bearer RcHoSkh8EH9UKo6iCHWXDpCKrmCyOIwr" -H "Content-Type:
 ↪  application/json" --data '{"courseId":"mbk-test-copied201910111508"}'
 ↪  https://partner-smoke-test-a.blackboard.com/learn/api/public/
v1/courses/_863_1/copy
```

```
{
  "status": 409,
  "message": "A course with the provided courseId already exists"
}
```

You can only use the copy endpoint to take an existing course and create and copy into a new, never before created, course.

ADD CONTENT

(Remember to Check Availability of the APIs in the Learn Version)

Preconditions - You know the courseId, you've gotten an access token, and you have a local file to upload.

1. Get a course's contents.

```
curl -k -X GET -H "Authorization: Bearer TwniVbrjLoQnNVWexAGBgQEyMaw7GTOP"
 ↪  https://bd-partner-a-original.blackboard.com/learn/api/public/v1/courses/
courseId:mbk-rest-contents/contents
```

```
{
  "results": [
    {
      "id": "_12906_1",
      "title": "Information",
      "created": "2017-12-06T18:50:24.625Z",
      "position": 1,
      "hasChildren": true,
      "availability": {
        "available": "Yes",
        "allowGuests": true,
        "adaptiveRelease": {}
      },
      "contentHandler": { "id": "resource/x-bb-folder" }
    },
    {
      "id": "**_12907_1**",
      "title": "Content",
      "created": "2017-12-06T18:50:24.628Z",
      "position": 2,
      "hasChildren": true,
      "availability": {
        "available": "Yes",
        "allowGuests": true,
        "adaptiveRelease": {}
      },
      "contentHandler": { "id": "resource/x-bb-folder" }
    }
  ]
}
```

In the THIRD step we'll place our file in the Content folder we see in the above response. Every new Learn Original course has a Content folder by default.

2. Upload a file.

```
curl -X POST -H "Authorization: Bearer TwniVbrjLoQnNVWexAGBgQEyMaw7GT0P"
 ↪  https://bd-partner-a-original.blackboard.com/learn/api/public/v1/uploads -F
 ↪  "file=@/Users/mbk/Documents/2016.09.BlackboardPartnerUpdate.pdf"
```

```
{ "id": "4B6281344DFDD9B1F36A5719BDB10708-38295ef0c6a74954a1055e1045bcfeeb" }
```

3. Create the content item using bbXML and the id from the prior step.

```
{
  "title": "Sept 2016 Partner Update",
  "contentHandler": {
    "id": "resource/x-bb-file",
    "file": {
      "uploadId":"4B6281344DFDD9B1F36A5719BDB10708-38295ef0c6a74954a1055e1045bcfeeb",
      "fileName": "2016.09.BlackboardPartnerUpdate.pdf",
      "duplicateFileHandling": "Rename"
    }
  }
}
```

Then call the appropriate endpoint:

```
curl -k -X POST -H "Authorization: Bearer TwniVbrjLoQnNVWexAGBgQEyMaw7GT0P" -H "Content-Type:
 ↪  application/json" --data '{"title": "Sept 2016 Partner Update", "contentHandler": {"id":
 ↪  "resource/x-bb-file", "file": {"uploadId":
 ↪  "4B6281344DFDD9B1F36A5719BDB10708-38295ef0c6a74954a1055e1045bcfeeb",
 ↪  "fileName":"2016.09.BlackboardPartnerUpdate.pdf", "duplicateFileHandling": "Rename"}}}'
 ↪  https://bd-partner-a-original.blackboard.com/learn/api/public/v1/courses/
courseId:mbk-rest-contents/contents/_12907_1/children
```

```
{
  "id": "_12908_1",
  "parentId": "_12907_1",
  "title": "Sept 2016 Partner Update",
  "created": "2017-12-06T19:22:59.536Z",
  "position": 0,
  "availability": {
    "available": "Yes",
    "allowGuests": true,
    "adaptiveRelease": {}
  },
  "contentHandler": {
    "id": "resource/x-bb-file",
    "file": { "fileName": "2016.09.BlackboardPartnerUpdate.pdf" }
  }
}
```

You can, of course, use the id returned in the above to fetch the content item using the appropriate GET. Read the documentation at developer.anthology.com.

CREATE ASSIGNMENT WITH ATTACHMENT

Currently Only Available For Ultra Courses (Remember to check the Learn version for availably of the APIs @ https://developer.anthology.com)

1. Get a course's content root. root will only work with the _abc_xyz ID format. It will not work for courseId: format.

```
curl -k -X GET -H "Authorization: Bearer <Access Token> "
 ↪  https://<LearnHost>/learn/api/public/v1/courses/<ID>/contents/root
```

Example:

```
curl -k -X GET -H "Authorization: Bearer N75W0ceKowxtlMUZtuIqwwvPQtxvx3L5"
 ↪  https://bd-partner-a-ultra.blackboard.com/learn/api/public/v1/courses/_691_1/contents/root
```


```
{
  "id": "_5108_1",
  "title": "ROOT",
  "created": "2018-04-08T15:57:34.188Z",
  "position": 0,
  "hasChildren": true,
  "availability": {
    "available": "Yes",
    "allowGuests": false,
    "adaptiveRelease": {}
  },
  "contentHandler": { "id": "resource/x-bb-folder" }
}
```


2. Upload the file to be attached to the assignment.

Note: The file is uploaded to temporary storage so we must attach it to an assignment shortly after the upload

```
curl -X POST -H "Authorization: Bearer N75W0ceKowxtlMUZtuIqwwvPQtxvx3L5"
 ↪  https://bd-partner-a-ultra.blackboard.com/learn/api/public/v1/uploads -F
 ↪  "file=@/Users/mbk/Documents/2016.06.BlackboardPartnerUpdate.pdf"
```


```
{ "id": "D1-D1165F42C7E11286BDAFDCCD2E2BE935-56c7b14aa4cb499b9a9a2a8cc3156290" }
```


We use this id to indicate the file that is to be attached to the assignment.

3. Create the assignment with the file attached:

(The file must be attached soon after being uploaded as it is uploaded to a temporary location and goes away sometime after being uploaded. Also, there is a bug in the following where if attempts allowed is too large it will fail. I've not experimented to find the limit.)

Reference: POST /learn/api/public/v1/courses/{courseId}/contents/createAssignment

Example:

```
curl -k -X POST -H "Authorization: Bearer N75W0ceKowxtlMUZtuIqwwvPQtxvx3L5" -H "Content-Type:
 ↪  application/json" --data '{"parentId": "_5108_1", "title":"Assignment Created by REST
 ↪  createAssignment", "instructions": "Simple Instructions", "description": "Assignment with
 ↪  Attachment", "position": 0,
 ↪  "fileUploadIds":["D1-D1165F42C7E11286BDAFDCCD2E2BE935-56c7b14aa4cb499b9a9a2a8cc3156290"],
 ↪  "availability": {"available": "Yes", "allowGuests": true, "adaptiveRelease": { "start":
 ↪  "2018-04-05T18:35:20.050Z", "end": "2018-09-02T18:35:20.050Z" } }, "grading": { "due":
 ↪  "2018-09-02T18:35:20.050Z", "attemptsAllowed": 10 }, "score": { "possible": 100 } }'
 ↪  https://bd-partner-a-ultra.blackboard.com/learn/api/public/v1/courses/
courseId:mbk-ultra-course/contents/createAssignment
```


```
{
  "contentId": "_6282_1",
  "gradeColumnId": "_3297_1",
  "assessmentId": "_11701_1",
  "questionIds": ["_11703_1", "_11704_1"]
}
```


We will use the gradeColumnId when submitting a student attempt to the assignment.

SUBMIT ASSIGNMENT ATTEMPT WITH ATTACHMENT

Currently Only Available For Presentation-Only Assignments

(Reference for all APIs used and the Learn version they are available in is @ https://developer.anthology.com)

1. Use 3LO to get an access code for the student account that will submit the attempt to the assignment. 3LO MUST be used for this process.

NOTES:

- Use Status read write, write is required to POST the assignment.
- LOG OUT of any browser session with the Learn system you've been working with before doing the next step where you will log in with the student account.

   a. Put the following URL in a browser's address field. Use the FQDN of the Learn system you are working with. The client_id is the REST Application Key. (Not the ID.)

```
https://bd-partner-a-ultra.blackboard.com/learn/api/public/v1/oauth2/authorizationcode?
↪ redirect_uri=https://localhost & response_type=code &
↪ client_id=d128e50d-c91e-47d3-a97e-9d0c8a77fb5d & scope=read write & state=xyzzy
```

Your browser should take you to the Learn login page. If you see the dialog to accept cookies, accept the dialog. Then repeat a. There is a bug that won't take you past b. if you've not previously accepted the cookie disclosure dialog.

   b. Log in with the student account that will be submitting the assignment attempt. Accept any dialog that pops up.

Your browser's address field will be redirected to a URL like the following. You SHOULD see something about being unreachable in the browser. That's OK.

```
https://localhost/?code=g84Xx0YaEz1iqS6HFRq0X9MdRTnQ48FT & state=xyzzy
```

Our code is: g84Xx0YaEz1iqS6HFRq0X9MdRTnQ48FT This is the code we use in the next step to get an access token.

2. Use the 2nd-leg of 3LO to get an access token - POST using the code, our application key, and secret to the oauth2/token endpoint.

Note: the redirect_uri must match the redirect_uri we used when we got the code.

```
curl -k --user d128e50d-c91e-47d3-a97e-9d0c8a77fb5d:sorryyoucanthavemysecret --data
↪ "grant_type=authorization_code"
↪ https://bd-partner-a-ultra.blackboard.com/learn/api/public/v1/oauth2/token?
↪ code=g84Xx0YaEz1iqS6HFRq0X9MdRTnQ48FT & redirect_uri=https://localhost
```

We get an access token back to use for submitting the students assignment attempt.

```
{
  "access_token": "4mgoFlQoi4Jq4biKpU4R264wugsKF9R1",
  "token_type": "bearer",
  "expires_in": 3599,
  "scope": "read write",
  "user_id": "507ba08376e1498cba8c6cd35b003aa2"
}
```

3. Upload the file to be attached to the attempt into temporary storage. It's temporary and needs to be used quickly

```
curl -X POST -H "Authorization: Bearer 4mgoFlQoi4Jq4biKpU4R264wugsKF9R1"
↪ https://bd-partner-a-ultra.blackboard.com/learn/api/public/v1/uploads -F
↪ "[file=@/Users/mbk/Documents/2016.03.BlackboardPartnerUpdate.pdf]"
```

We get back an ID that we use to attach the file to the attempt.

```
{ "id": "BD-BD9C08679892561211D99DB4C817FE68-67545ce854b54d3fb062e38aefeee47c" }
```

4. Submit the attempt. We're using the gradeColumnId":"_3297_1" that we got when we created the assignment.

The "studentSubmision" is in bbML format. Reference: Blackboard Markup Language - BbML

```
curl -k -X POST -H "Authorization: Bearer 4mgoFlQoi4Jq4biKpU4R264wugsKF9R1" -H "Content-Type:
 ↪  application/json" --data '{"studentComments":"this is the student commment",
 ↪  "studentSubmission": "<!-- {\"bbMLEditorVersion\":1} --> <a
 ↪  href=\"bbupload://BD-BD9C08679892561211D99DB4C817FE68-67545ce854b54d3fb062e38aefeee47c\"
 ↪  data-bbfile=\"{ & quot;render & quot;: & quot;inline & quot;, & quot;linkName & quot;: &
 ↪  quot;2016.03.BlackboardPartnerUpdate.pdf & quot;, & quot;mimeType & quot;: &
 ↪  quot;application/pdf & quot;}\">2016.03.BlackboardPartnerUpdate.pdf<_/a>"}:_'
 ↪  https://bd-partner-a-ultra.blackboard.com/learn/api/public/v2/courses/courseId:mbk-ultra-
course/gradebook/columns/_3297_1/attempts
```

We get back an attempt ID for an InProgress student submission. We use the attempt ID in the next step to set the attempt to needs grading so that the instructor sees the attempt. Attempts that are InProgress are not available to the instructor in the GUI.

```
{
  "id": "_528_1",
  "userId": "_649_1",
  "status": "InProgress",
  "studentSubmission": "<!-- {\"bbMLEditorVersion\":1} -->\n<a
   ↪  href=\"https://bd-partner-a-ultra.blackboard.com/bbcswebdav/xid-23611_1\" data-bbfile=\"{
   ↪  & quot;render & quot;: & quot;inline & quot;, & quot;linkName & quot;: &
   ↪  quot;2016.03.BlackboardPartnerUpdate.pdf & quot;, & quot;mimeType & quot;: &
   ↪  quot;application/pdf & quot;}\">2016.03.BlackboardPartnerUpdate.pdf</a>",
  "exempt": false,
  "created": "2018-05-18T02:48:18.329Z"
}
```

5. Set the attempt to "NeedsGrading"

```
curl -k -X PATCH -H "Authorization: Bearer 4mgoFlQoi4Jq4biKpU4R264wugsKF9R1" -H "Content-Type:
 ↪  application/json" --data '{"status":"NeedsGrading"}:'
 ↪  https://bd-partner-a-ultra.blackboard.com/learn/api/public/v2/courses/courseId:mbk-ultra-
course/gradebook/columns/_3297_1/attempts_528_1
```

```
{"id":"528_1", "userId":"_649_1", "status":"NeedsGrading", "studentSubmission":"<!--
 ↪  {\"bbMLEditorVersion\":1} -->\n<a
 ↪  href=\"[https://bd-partner-a-ultra.blackboard.com/bbcswebdav/xid-23611_1" data-bbfile=\"{ &
 ↪  quot;render & quot;: & quot;inline & quot;, & quot;linkName & quot;: &
 ↪  quot;2016.03.BlackboardPartnerUpdate.pdf & quot;, & quot;mimeType & quot;: &
 ↪  quot;application/pdf & quot;}\">2016.03.BlackboardPartnerUpdate.pdf</a>", "exempt":false,
 ↪  "created":"2018-05-18T02:48:18.329Z"}
```

{: .code-text-to-normal}

Now, logged in as the instructor, we can see that there has been 1 submission. And we see the student's submission with the attachment.

Common Errors

- COMMON ERROR: You used 3LO to get an Access Token. You forgot to set the scope correctly and are getting permission denied errors. 404 etc.
- SOLUTION: Go back through the 3LO process and specify the scope necessary to make your REST calls.

Conclusion

We hope that the above demonstration gives you a helpful peek under the hood as to what is going on when you make REST calls to a Learn system. You can take these cURL commands and use them to make all of the REST calls documented here: Explore APIs