API Rate Limits in Learning Manager

Introduction to rate limiting {#introductiontoratelimiting}

Adobe Learning Manager exposes a rich suite of REST API that helps customers build applications that integrate with Learning Manager, or even custom user experiences and extensions to workflows that help their business.

Whenever an API is called there are shared computational resources that get utilized in the Learning Manager servers, and so we must exercise care and caution to ensure that the applications play nice and don't jeopardize the performance and availability characteristics of the platform. This is accomplished by introducing limits on how API clients make calls (frequency and velocity).

For example, an application could be trying to get all users in that account and may make multiple paginated API calls at a fast rate. And by mistake, a developer could call this in a tight loop resulting in a huge load on the server and making applications sluggish and even endangering the platform by consuming more resources than what is intended or required.

To solve this problem, we are introducing rate limits on how many API calls can be made by a single user in a specific client application of a specific account. We measure this in terms of Requests Per Minute, abbreviated as RPM. For example, when we say the limit for GET API calls is 600 RPM, it simply means that a single user cannot exceed a maximum of 600 calls in a minute, and if the user exceeds that limit the user will get rate limited. The requests are tracked at millisecond granularity, so this limit corresponds to 1 request every 100 milliseconds (ms). There is one more parameter, called Burst, which is also defined along with rate limit, which defines how many requests a user can make in excess of the rate specified (in our case, 600RPM). For example, if the Burst parameter is 10; it means that up to 10 slots will be allocated in a queue and when a request arrives "too soon" (in our case, sooner than 100ms), it is processed immediately if there is a slot available for it in the queue. The slot is then marked as "taken" and is not freed up for use by another request until the appropriate time has passed (in our case, after 100ms). Real world traffic is generally with bursts, and that is where the Burst parameter helps. For the Learning Manager platform, we define limits for a specific API version (say V2), role (Learner / Admin) and a verb (GET/PUT/POST/DELETE/PATCH). In the example, describe above we would say the rate limit is (600, 10).

Although we have always had rate limits for Public API in Learning Manager; we have been quite liberal in allowing a very high RPM so far. However, with the growth of your favorite learning platform, we have seen rapid growth in the usage of our API, and we have decided to make these rate limits be explicitly documented for the benefit of all our customers and for better management of the platform. In this context, we have upgraded our API to start returning a new API response (HTTP Status Code 429); which you have not seen so far. This response is provided to API clients, when the rate limit is breached. Client applications will now need to handle this response code as befitting the logic of their application; and this document is primarily aimed at helping developers with the information they need for incorporating the right logic in their code when they see such a response.

How to confirm the enforced rate limits? {#howtoconfirmtheenforcedratelimits}

For each request, the response headers contain the following information:

```
x-rate-limit: 600r/m
x-burst: 10
```

- x-rate-limit specifies the base request rate threshold in terms of requests per minute.
- x-burst specifies how many exceeding requests above the base request rate are accepted to be processed immediately.

How to catch errors caused by rate limits? {#howtocatcherrorscausedbyratelimits}

For each request, you can check to see if you have bumped into the rate limit. If you get a response code of 429, "{"message":"429 Too many requests"}", you have hit the rate limit.

It is best practice to include code in your script that catches 429 responses. If your script ignores the "Too many requests" error and keeps trying to make requests, you might start getting null errors. At that point, the error information won't be useful in diagnosing the problem.

For example, a curl request that bumps into the rate limit might return the following response:

```
< HTTP/2 429
< date: Wed, 04 Nov 2020 06:53:04 GMT
< content-type: application/json; charset=utf-8
< server: openresty
< x-rate-limit: 60r/m
< x-burst: 10
< retry-after: 10.752
< access-control-allow-credentials: true
< access-control-allow-methods: GET, POST, OPTIONS, PUT, HEAD, DELETE,
PATCH
< access-control-allow-headers: X-acap-all-roles, X-acap-account,X-acap-
user,X-acap-caller-role,Keep-Alive,User-Agent,X-Requested-With,If-
Modified-Since,Cache-Control,Content-Type, x-experience-api-version,
Authorization, X-CSRF-TOKEN, X-HTTP-Method-Override
< strict-transport-security: max-age=31536000; includeSubDomains
< x-frame-options: SAMEORIGIN
< x-xss-protection: 1
< x-content-type-options: nosniff
< x-request-id: gwBBFC9741-58A5-43B1-B1FE-7D14275961E7
< strict-transport-security: max-age=31536000; includeSubDomains
```

The response contains information about the following keys:

```
status: 429
retry-after: 10.752
```

The status code of 429 means "too many requests" and that the current request isn't processed. The retry-after header specifies that you can retry the API call in 10.752 seconds. Your code should stop making additional API requests until enough time has passed to retry.

The following pseudo-code shows a simple way to catch rate-limit errors:

```
response = request.get(url)
if response.status equals 429:
    alert('Rate limited. Waiting to retry…')
    wait(response.retry-after)
    retry(url)
```

In fact, we've even created a Learning Manager dummy API for you to play around and get comfortable with handling the 429 status code.

```
curl -X GET --header 'Accept: application/json' --header 'Authorization:
oauth <
<token>
  >' 'https://learningmanager.adobe.com/learningmanagerapi/v2/dummy
</token>
```

This dummy API has a limit of:

```
x-rate-limit: 5r/m
x-burst: 2
```

The limit for the dummy API is intentionally low so that you bump into the rate limits very quickly and can then focus more on developing the code to catch and handle the rate limit errors.

Testing the dummy API {#testingthedummyapi}

We have provided an endpoint for you to check out how rate limiting works. For this we have created a special endpoint called GET /dummy which has learner read scope. This API has deliberately been configured for a very strict rate limit of 5 requests per minute, with burst limit = 2.

You can easily test this out by hitting this endpoint with rates below 5 RPM and above 5 RPM. Write code to handle the HTTPS status code of 429 and based on the information in response headers.

To make it easy for you can check out this sample JavaCcript code which illustrates this. Click this fiddle and see the code in action.

This application requires you to provide a learner role application token for your account. Please refer to [Application Developer Manual](https://captivateLearning Manager.adobe.com/docs/Learning Managerapi/v2/) for information on API tokens and you can use the Token Helper in the developer resources section of the Learning Manager Integration Admin application to generate the tokens.

This application is making 10 calls to the dummy API in a loop, in one go. Since the rate limit is (5, 2) for the dummy API; the rate limit will be breached after the first 5+2 calls that are received by Learning Manager will succeed and you will see a success response for them.

However, note how this application is looking for 429 status code in response and handling them. When this application gets such a response, it prints out the details in the API response, waits for suggested amount of time and retries the failed attempts.

Best practices to handle rate limiting {#bestpracticestohandleratelimiting}

Many a time, data in an account does not change that often. For example, the courses in an account may not change that often. Instead of trying to get all data every time, see if you can get the specific data when you need them and consider using a cache mechanism. This way, when your application really needs to make many calls, you will have sufficient quota within your rate limits to make those important calls.

Some data may change more often, and you may need to get it more often. Even if so, ask yourself if your application can afford to have slightly stale data (something that may be in your cache, which you fetched N minutes ago), because it may have no bearing on the user's workflow. Even if you must get fresh data from the servers, you can again be judicious in fetching just the specific data you need, instead of getting everything.

There will also be times, when you have no choice to get a lot of data because the API may not be flexible enough to give you exactly what you want with just one call. See if the API provides you with filters and sort criteria that can be used to minimize the number of calls; and you should still consider caching because many users in your account may need the same data.

When you are getting too much of data in the context of an interactive application with a GUI, it is likely that the application is trying to do too much and could be overwhelming the user with a lot of information/functionality impacting the user experience of your application.

When you are building a non-interactive application (perhaps something like a daily job) you may need to fetch a lot of data in bulk. In such cases consider using the Job API which are primarily designed to return bulk data in CSV format. Note that Job API will have a quota constraint that you could invoke them N times a day.

Since Learning Manager implemented the JSONAPI specifications, there are API where you can also sideload some data. This will help you to conserve making additional API calls, but you will have to trade this off with getting data too eagerly. Since the sideloaded data can sometimes be very large, in API paginated calls you make the max page size is limited to 10; but for API calls without includes you may be able to specify a larger page size (up to 100)

Eventually, if you make a lot of API requests in a short amount of time, you will bump into the API rate limit for requests. When you reach the limit, Learning Manager will stop processing any more requests until a certain amount of time has passed.

The rate limits are outlined in the last section of this article. It is recommended for you to get familiar with the limits.

Rate Limits for V2 API Endpoints {#ratelimitsforv2apiendpoints}

The following table provides the limits for the various V2 endpoints. At present, the rate limits are not being imposed on customers, but these limits will be enforced from DD/MM/YYYY. And therefore, it becomes important for customers to review their existing applications' code to see how they can adjust their code to be aware of these rate limits and handle the API responses with HTTP status code of 429.

Verb	Admin API	Learner API
DELETE	(25, 10)	(20, 10)
PATCH	(60, 20)	(15, 5)
POST	(30, 10)	(30, 10)
PUT	(20, 10)	(20, 10)
GET	(100, 100)	(100, 30)