Get Interpretable Values with the Metadata API

The Monetate Metadata API allows you to retrieve interpretable values for experiences, page events, and custom targets. Monetate provides unique IDs because they never change, whereas you can change the name of experience variants and page events in the platform. You can send a request to the API with an experience ID, page event ID, or target ID to return a set of interpretable values.

Field	Type	Description
Custom target ID	Integer	Monetate's internal identifier for custom targets
Custom target title	String	The custom target's title
Custom target description	String	The custom target's description
Page event ID	Integer	Monetate's internal identifier for page events
Page event title	String	The page event's title
Experience ID	Integer	Monetate's internal identifier for experiences
Experience name	String	The experience's name
Account domain	String	The account domain the experience is associated with
Split letter	String	The letter associated with the variant for this experience
Experience type	String	The type of experience (Dynamic Testing, Automated Personalization, etc.)

Response Codes

When you make a request to the Monetate Metadata API, Monetate returns a response code and message. The code and message can vary based on the request and can help diagnose failures, invalid requests, and errors.

- **401 Unauthorized** — The request didn't include a token, or the token provided is invalid. Ensure that the token is correct and that the authorization header is properly formatted.
- **403 Forbidden** — The request included a token that's been revoked. Contact your account administrator to generate a new token.
- **404 Not found** — The resource you're trying to fetch doesn't exist or has been deleted.
- **500 Unknown error** — Try again or contact your account manager for more information.

Access and References

Your application needs to identify itself every time it sends a request to the Monetate Metadata API. The Monetate Auth API allows you to manage tokens on a per-user key basis. With this API, you can request a token that has a fixed expiration, that is associated with a public key you uploaded to Monetate, and that may be revoked by deactivating that key. Refer to the API documentation in the platform for additional information.

Monetate Metadata API Code Demo

This example script, in version 3 of Python, demonstrates how to look up metadata for unique IDs found in Session Stream.

```
# coding: utf-8

from collections import Counter

import json

import jwt

import requests

import time

# We have here a sample session data file. You can download one following the documentation for

with open('one_hour.json', 'r') as f:
  raw_data = f.read()

sessions = [json.loads(line) for line in raw_data.splitlines()]

# The session data in this file refers to objects in the platform like experiences, page events, and t
# But to review this data, we want to turn these IDs into human-interpretable names. The Metadat
# Metadata API requires authentication. Let's first get an auth token.
# - Send a GET request to the refresh endpoint with an 'Authorization' header.
# - The value of the header must be 'JWT <token>' where <token> is an encoded JWT token as de
# - The token's payload must contain an 'iat' claim and a 'username' claim.
# - The 'iat' claim must contain a Unix timestamp of when the request is sent. A leeway of 60 secor
# - The API username used for the 'username' claim must have a public key configured in the UI (o
# - The private key used to encode the JWT token must be the matching private key for the aforem

# Create the payload. In the 'username' claim, use the username that is generated by the API Key

payload = {'iat': int(time.time()), 'username': 'api-111-demo'}

# Load up your private key and use it to encode the payload. We allow RS256, RS384, and RS512 a

with open('private.pem', 'r') as f:
  private_key = f.read()

jwt_token = jwt.encode(payload, private_key, algorithm='RS256')

# Assemble and send the request to the auth endpoint.
# The default TTL for the returned auth token is 3600 seconds (1 hour).
```

```
# You may optionally send a 'ttl' query parameter to request a specific TTL in seconds, between 60

AUTH_REFRESH_URL = 'https://api.monetate.net/api/auth/v0/refresh/'

resp = requests.get(AUTH_REFRESH_URL,
  headers = {'Authorization': 'JWT {}'.format(jwt_token)})

# Grab the token from the response. Note that the response also contains an expiration timestamp

auth_token = resp.json()['data']['token']

# Now that we have a token, we can use it to access the Metadata API.
# Replace "example" with your retailer shortname found in the base URL of your API documentatio

METADATA_BASE_URL = \
  'https://api.monetate.net/api/metadata/v1/example/production/metadata/'

PAGE_EVENT_URL = METADATA_BASE_URL + 'pageevent/'

VARIANT_URL = METADATA_BASE_URL + 'variant/'

TARGET_URL = METADATA_BASE_URL + 'customtarget/'

# Let's go ahead and fetch a bunch of page events. 1000 is the maximum allowed in one request.
# These are paginated, so you can make smaller requests and piece them together later.

page_event_resp = requests.get(PAGE_EVENT_URL,
  headers = {'Authorization': 'Token {}'.format(auth_token)},
  params = {'page_size': 1000})

# Grab the list of events. Make it a dict so we can reference by ID.

page_event_list = page_event_resp.json()['data']

page_event_data = {d['id']: d for d in page_event_list}

# Now let's marry this data to your session data from earlier.
# Session data has a property page_event_ids which is a dict of page event IDs -> the time at whic
# These page event IDs are the same ones retrieved in the last Metadata API response.
# Let's loop over the sessions and print the city the user browsed in and triggered page events for

missing_page_events = []

for session_data in sessions:
  city = session_data['city']
  session_page_events = []
```

```
 for event_id in session_data['page_event_ids']:

# We'll need to coerce the IDs back into ints since JSON forces dict keys to be strings.

 try:
   page_event_metadata = page_event_data[int(event_id)]

# Let's grab the title

   session_page_events.append(page_event_metadata['title'])
    except KeyError:
    missing_page_events.append(event_id)

print('A user in {} triggered the following page events: {}'.format(city, session_page_events))

print('{} page events could not be matched; do you need to request more page event definitions fr

# Now let's grab experience data. Here, we'll also show how to do some simple pagination.

has_next = True

variant_list = []

page = 1

while has_next:
  variant_resp = requests.get(VARIANT_URL,
    headers = {'Authorization': 'Token {}'.format(auth_token)},
    params = {'page_size': 1000, 'page': page})

    parsed_response = variant_resp.json()

# Grab the list of experiences from this page.

  variant_list.extend(parsed_response['data'])

# Are there more pages to fetch?

  if parsed_response['meta'].get('next', None):
    page = page + 1
  else:
    has_next = False

# Make it a dict so we can reference by ID.

variant_data = {e['id']: e for e in variant_list}
```

```
# offer_ids in our session data contain experience IDs, which we've just fetched definitions for from
# We'll do a simple counter to see what experiences were seen in the given day's worth of data.

experiences_shown = []

for session in sessions:

  for offer_id in session['offers']:

# Offer IDs have an extra piece at the end indicating the variant, which you'll want to discard for o

    experience_id = offer_id.split('-')[0]

experiences_shown.append(experience_id)

experience_counter = Counter(experiences_shown)

# Now we can put the two together. Let's take the top 10 most seen experiences and display their

top_10 = experience_counter.most_common(10)

for experience_id, count in top_10:

  print("{} was seen by {} sessions.".format(variant_data[experience_id]['experience_name'], coun

# Let's also fetch custom target metadata. Simply follow the same pattern.

target_resp = requests.get(TARGET_URL,
  headers = {'Authorization': 'Token {}'.format(auth_token)},
  params = {'page_size': 1000})

# Grab the list of targets.

target_list = page_event_resp.json()['data']

# Now let's make it a dict so we can reference by ID.

target_data = {d['id']: d for d in target_list}

# The above dict is keyed on custom target IDs, which are present as keys in the custom_targets d
# You can marry the two just like we did above to do any of your own analytics.
```