Instagram Graph API Guide

# Core Setup Pattern

All scripts follow this authentication pattern:  
  
```python  
import requests, os  
from dotenv import load\_dotenv  
load\_dotenv()  
access\_token = os.getenv('ACCESS\_TOKEN')  
```  
  
How it works:  
  
- load\_dotenv() loads environment variables from a .env file.  
- ACCESS\_TOKEN should be stored in .env:  
  
```text  
ACCESS\_TOKEN=your\_token\_here  
```  
  
This keeps your sensitive token out of your code.

# API Request Structure

Standard request format for all endpoints:  
  
```python  
response = requests.get(url, params={  
 'fields': 'field1,field2,field3',  
 'access\_token': access\_token  
})  
if response.status\_code == 200:  
 data = response.json().get('data', [])  
 # Process data  
else:  
 print(f"Error: {response.status\_code}")  
```

# Key Endpoints & Use Cases

1. User Profile (/me)  
Purpose: Test connection and get basic user info  
  
Example:  
  
```python  
url = 'https://graph.instagram.com/me'  
fields = 'user\_id,username'  
```  
  
2. User Media (/me/media)  
Purpose: Fetch all posts from user's account  
  
Example:  
  
```python  
url = 'https://graph.instagram.com/me/media'  
fields = 'id,media\_type,media\_product\_type,media\_url,caption,timestamp,like\_count,comments\_count,permalink,tags,children'  
```  
  
Additional Fields:  
- media\_product\_type: Distinguishes Reels (REELS) from feed posts (FEED)  
- tags: Tagged usernames  
- children: For carousels  
- permalink: Direct post link  
  
3. Post Comments (/{media\_id}/comments)  
Purpose: Get comments for a specific post  
  
Example:  
  
```python  
url = f'https://graph.instagram.com/{media\_id}/comments'  
fields = 'text,username,timestamp,replies'  
```  
  
Additional Fields:  
- replies: Threaded comment replies  
  
4. Media Insights (/{media\_id}/insights)  
Purpose: Fetch engagement/performance metrics (Business/Creator accounts only)  
  
Example:  
  
```python  
url = f'https://graph.instagram.com/{media\_id}/insights'  
fields = 'reach,likes,comments,saved,shares,total\_interactions,profile\_visits,follows,profile\_activity' # For photos/carousels  
fields = 'reach,likes,comments,saved,shares,views,ig\_reels\_video\_view\_total\_time,ig\_reels\_avg\_watch\_time,total\_interactions,profile\_visits,follows,profile\_activity' # For Reels  
```  
  
How it works:  
- Requires instagram\_business\_manage\_insights permission  
- Metrics vary by media type and product type  
- Reels-specific: views, ig\_reels\_video\_view\_total\_time, ig\_reels\_avg\_watch\_time  
  
5. Create Media Container (/me/media - POST)  
Purpose: Prepare an image/video for posting  
  
Example:  
  
```python  
url = 'https://graph.instagram.com/me/media'  
params = {  
 'image\_url': 'https://example.com/image.jpg',  
 'caption': 'Your caption text #hashtags',  
 'access\_token': access\_token  
}  
response = requests.post(url, params=params)  
```  
  
6. Publish Media (/me/media\_publish - POST)  
Purpose: Publish the prepared media  
  
Example:  
  
```python  
url = 'https://graph.instagram.com/me/media\_publish'  
params = {  
 'creation\_id': container\_id,  
 'access\_token': access\_token  
}  
response = requests.post(url, params=params)  
```  
  
7. Add Comment (/{media\_id}/comments - POST)  
Purpose: Add a comment to a specific post  
  
Example:  
  
```python  
url = f'https://graph.instagram.com/{media\_id}/comments'  
params = {  
 'message': comment\_text,  
 'access\_token': access\_token  
}  
response = requests.post(url, params=params)  
```  
  
8. Complete Account Analysis (everything.py)  
Purpose: Comprehensive Instagram analytics for campaign insights  
  
Features:  
- Fetch user info, all media, comments, insights  
- Analyze engagement: likes, comments, reach, saved, shares, views  
- Detect Reels vs Feed  
- Generate dashboard  
- Compute engagement rate  
- Format watch time metrics  
- Suggest content strategies  
  
```python  
from everything import analyze\_instagram\_account  
analyze\_instagram\_account()  
```

# Data Processing Pattern

Nested API Calls:  
  
```python  
for media in media\_data:  
 media\_id = media['id']  
 comments\_response = requests.get(comments\_url, params=comments\_params)  
 insights\_response = requests.get(insights\_url, params=insights\_params)  
 comments\_data = comments\_response.json().get('data', [])  
 insights\_data = insights\_response.json().get('data', [])  
```  
  
Two-Step Publishing:  
  
Step 1: Container Creation  
- Validate content  
- Return container\_id  
  
Step 2: Media Publishing  
- Uses container\_id to post  
- Returns media\_id

# Common Field Options

|  |  |
| --- | --- |
| Endpoint | Fields |
| /me | user\_id, username |
| /me/media | id, media\_type, media\_product\_type, media\_url, caption, timestamp, like\_count, comments\_count, permalink, tags, children |
| /{id}/comments | text, username, timestamp, replies |
| /{id}/insights | reach, likes, comments, saved, shares, views, ig\_reels\_video\_view\_total\_time, ig\_reels\_avg\_watch\_time, total\_interactions, profile\_visits, follows, profile\_activity |

# Error Handling Best Practice

```python  
if response.status\_code == 200:  
 data = response.json()  
 print(f"Success: {data}")  
else:  
 print(f"Error {response.status\_code}: {response.text}")  
```

# Important Requirements & Limitations

Image Requirements  
- Must be publicly accessible HTTPS URL  
- Formats: JPG, PNG  
- Max size: 8MB  
- No auth required for access  
  
Caption Requirements  
- Max 2,200 characters  
- Hashtags count  
- Line breaks allowed using \n  
  
Access Token  
- Tokens expire  
- Requires proper permissions  
- Test with /me endpoint  
  
Rate Limiting  
- Instagram enforces hourly limits  
- Add delays for bulk tasks  
- Monitor status codes