LinkedIn `/v2/userinfo` — End-to-End POC (Python)

This doc shows the **exact process** the team used to test LinkedIn's official **OpenID Connect userinfo** endpoint with Python. It's designed so others can follow the same pattern for future APIs.

---

1) What you'll build

- A **token-setter** script that launches the LinkedIn consent screen, captures the `code`, exchanges it for an **access token**, and **sets `LI_TOKEN` in your shell**.
- A tiny **caller** script that reads `LI_TOKEN` and calls `https://api.linkedin.com/v2/userinfo`

  Works on macOS/Linux shells; Windows PowerShell users can adapt the `eval`/`export` bits.

---

2) Prerequisites

- A LinkedIn Developer App with the **Sign in with LinkedIn using OpenID Connect** product.
- In the app's **Auth** → **Redirect URLs**, add: http://localhost:8000/callback
- Your app's **Client ID** and **Client Secret**.
- Python 3.9+ and `pip`.

---

3) Project files

Create two files in a folder (e.g., `linkedin-app/`):

- `linkedin_token_setter.py` – fetches a token and prints an `export LI_TOKEN='...'` line.
- `use_li_token.py` – calls `/v2/userinfo` using the token.

---

4) Install dependencies

```
python3 -m pip install requests
```

---

5) Get and set the access token (one-time per ~60 days)

Run the **token setter**. It opens the consent page, captures the redirect, exchanges the code, **sets `LI_TOKEN` in your current shell**, and prints the **exact local expiry timestamp**.

```
eval "$(python3 linkedin_token_setter.py \

--client-id '<YOUR_CLIENT_ID>' \

--client-secret '<YOUR_CLIENT_SECRET>' \

--redirect-uri 'http://localhost:8000/callback' \

--scopes 'openid profile')"
```

What you’ll see (example):

```
[INFO] Browser opened for LinkedIn consent. Waiting for redirect...

[INFO] Access token fetched successfully!

[INFO] Valid for: 5,183,999 seconds (~60.0 days)

[INFO] Expires on: 2025-12-03 13:57:20 EDT (-0400)
```

The “Expires on” line is the **exact local date/time** when the token becomes invalid. Copy this into your notes/calendar so you know when to refresh.

Verify the token is set:

```
echo "$LI_TOKEN"
```

(You should see a long string; don’t share it.)

Notes

- `openid profile` is sufficient for `/v2/userinfo`. Add `email` if you also want the email returned (rerun the command with `--scopes 'openid profile email'` to mint a new token).
- The **authorization code** (used internally) is single-use and short-lived.
- The **access token** lasts until the printed **Expires on** timestamp (≈ 60 days).
- When the token expires, **rerun the command above** to generate a fresh one and reset `LI_TOKEN`.

---

6) Call the official API

```
LI_TOKEN="$LI_TOKEN" python3 use_li_token.py
```

Expected output (example):

```
{
  "name": "Your Name",
  "sub": "BtXn4sJ7GQ",
  "locale": {"country": "US", "language": "en"},
  "given_name": "Your",
  "family_name": "Name",
  "picture": "https://media.licdn.com/..."
}
```

---

7) Adapting this pattern for other LinkedIn APIs

1. **Decide endpoint + scopes** you need (e.g., email → add `email`; org posts → different app with Community Management and `r_organization_social`).
2. **Re-run the token setter** with the required scopes.
   Email example:

   ```
    eval "$(python3 linkedin_token_setter.py \

   --client-id '<YOUR_CLIENT_ID>' \
   --client-secret '<YOUR_CLIENT_SECRET>' \
   --redirect-uri 'http://localhost:8000/callback' \
   --scopes 'openid profile email')"
   ```

3. **Call the new endpoint** from a Python script using the same `LI_TOKEN`.

   For classic REST endpoints (e.g., Posts), include LinkedIn REST headers:

   ```
    headers = {
     "Authorization": f"Bearer {LI_TOKEN}",
     "X-Restli-Protocol-Version": "2.0.0",
     "LinkedIn-Version": "202509"  # pin a modern YYYYMM
   }
   ```

---

8) Token renewal

- `LI_TOKEN` (access token) lasts ~**60 days**.
- When it expires (you'll get `401 Unauthorized`), **re-run the token setter** to mint a fresh token and set `LI_TOKEN` again. No code changes required.

---

9) Troubleshooting quick refs

- **Browser says redirect URI not allowed**
  Add **`http://localhost:8000/callback`** in your app's **Redirect URLs** and use the same in the command.

- **`invalid_client` on token exchange**
  Client ID/secret mismatch or redirect mismatch. Re-copy both and ensure the redirect matches exactly.

- **`openid_insufficient_scope_error`**
  You asked for `openid` but didn't include a `profile` (or `email` when needed). Use `--scopes 'openid profile'` (and add `email` if needed).

- **Token is empty in Python**
  You didn't run the setter with `eval "$( ... )"`. The export line must be evaluated by your shell to set `LI_TOKEN`.

---

10) Security & hygiene

- **Never commit** client secrets or tokens to version control.
- Prefer storing secrets in your vault; for local dev, environment variables are OK.
- Treat `LI_TOKEN` like a password (it grants API access until expiry).

Quick Summary — *Example token-setter command (team reference)*

Replace with your own IDs in your copy of this doc.

```
eval "$(python3 linkedin_token_setter.py \
--client-id '78seural244bqk' \
--client-secret 'WPL_AP1.FrsC8ykRzecvEHWx.us0QDQ==' \
--redirect-uri 'http://localhost:8000/callback' \
--scopes 'openid profile')"
```

Then:

```
echo "$LI_TOKEN"              # sanity check

LI_TOKEN="$LI_TOKEN" python3 use_li_token.py
```
