# Python SDK Integration Tutorial

Complete guide to building production-ready Monzo integrations

**Version:** 1.0

**Author:** Technical Documentation Team

**Category:** API Integration | Python SDK

**Last Updated:** December 11, 2025

This comprehensive guide walks you through integrating the Monzo API into your Python application using best practices and production-ready patterns. While Monzo provides excellent API documentation, they do not maintain official SDKs. This tutorial bridges that gap with a complete implementation guide.

## Prerequisites

Before beginning, ensure you have:

• **Python 3.8+** installed on your system

• A Monzo developer account registered at **developers.monzo.com**

• OAuth credentials (Client ID and Client Secret) for your application

• Basic understanding of REST APIs and OAuth 2.0 authentication

• Familiarity with Python async/await patterns (for advanced implementations)

## Installation and Dependencies

Start by creating a clean Python environment and installing required packages:

```
# Create virtual environment python -m venv monzo_env source monzo_env/bin/activate # On
Windows: monzo_env\Scripts\activate # Install dependencies pip install requests
python-dotenv # For production applications, also install: pip install redis httpx tenacity
```

■ ***NOTE:*** *The requests library handles HTTP communication, python-dotenv manages environment variables, httpx enables async requests, and tenacity provides retry logic for production resilience.*

## Environment Configuration

Store your Monzo API credentials securely using environment variables. Never commit credentials to version control.

### Create .env file

```
# .env file (add to .gitignore!) MONZO_CLIENT_ID=oauth2client_00009A1BK4MH...
MONZO_CLIENT_SECRET=mnzconf.Z7pBfXE2rD...
MONZO_REDIRECT_URI=http://localhost:8080/callback
MONZO_STATE_TOKEN=randomly_generated_string_for_csrf_protection
```

### Load configuration in Python

```
import os from dotenv import load_dotenv load_dotenv() class MonzoConfig: """Monzo API
configuration""" CLIENT_ID = os.getenv('MONZO_CLIENT_ID') CLIENT_SECRET =
os.getenv('MONZO_CLIENT_SECRET') REDIRECT_URI = os.getenv('MONZO_REDIRECT_URI')
STATE_TOKEN = os.getenv('MONZO_STATE_TOKEN') # API endpoints BASE_URL =
'https://api.monzo.com' AUTH_URL = 'https://auth.monzo.com' @classmethod def
validate(cls): """Ensure all required config is present""" required = ['CLIENT_ID',
'CLIENT_SECRET', 'REDIRECT_URI'] missing = [key for key in required if not getattr(cls,
key)] if missing: raise ValueError(f"Missing config: {', '.join(missing)}")
```

## OAuth 2.0 Authentication Flow

Monzo uses OAuth 2.0 authorization code flow. This requires user authorization through Monzo's web interface before your application can access their account data.

### Step 1: Generate Authorization URL

```
import urllib.parse def get_authorization_url(config): """ Generate OAuth authorization URL
for user to visit. User will be redirected to your REDIRECT_URI with an authorization code.
""" params = { 'client_id': config.CLIENT_ID, 'redirect_uri': config.REDIRECT_URI,
'response_type': 'code', 'state': config.STATE_TOKEN # CSRF protection } auth_url =
f"{config.AUTH_URL}/?{urllib.parse.urlencode(params)}" return auth_url # Usage
MonzoConfig.validate() auth_url = get_authorization_url(MonzoConfig) print(f"Visit this
URL to authorize:\n{auth_url}")
```

### Step 2: Exchange Authorization Code for Access Token

After the user authorizes your application, Monzo redirects them to your callback URL with an authorization code. Exchange this code for an access token:

```
import requests def exchange_auth_code(auth_code, config): """ Exchange authorization code
for access token. Args: auth_code: Code received from OAuth callback config: MonzoConfig
instance Returns: dict: Token response containing access_token, refresh_token, etc. """
token_url = f"{config.BASE_URL}/oauth2/token" data = { 'grant_type': 'authorization_code',
'client_id': config.CLIENT_ID, 'client_secret': config.CLIENT_SECRET, 'redirect_uri':
config.REDIRECT_URI, 'code': auth_code } response = requests.post(token_url, data=data)
response.raise_for_status() return response.json() # Example response: # { #
"access_token": "eyJhbGciOiJFUzI1NiIsInR5cCI6...", # "token_type": "Bearer", #
"expires_in": 21600, # 6 hours # "refresh_token": "eyJhbGciOiJFUzI1NiIsInR5cCI6...", #
"user_id": "user_00009238aq...", # "client_id": "oauth2client_00009A1BK..." # }
```

**■■ WARNING: Access tokens expire after 6 hours. Always implement token refresh logic for production applications to maintain continuous API access.**

### Step 3: Refresh Access Token

```
def refresh_access_token(refresh_token, config): """ Obtain new access token using refresh
token. Call this when access_token expires (every 6 hours). """ token_url =
f"{config.BASE_URL}/oauth2/token" data = { 'grant_type': 'refresh_token', 'client_id':
config.CLIENT_ID, 'client_secret': config.CLIENT_SECRET, 'refresh_token': refresh_token }
response = requests.post(token_url, data=data) response.raise_for_status() return
```

```
response.json()
```

# Building the Monzo Client Class

Create a reusable client class that handles authentication, token management, and API requests:

```
import time from typing import Optional, Dict, List class MonzoClient: """Production-ready Monzo API client""" def __init__(self, access_token: str, refresh_token: Optional[str] = None, expires_at: Optional[float] = None, config=None): """ Initialize Monzo client. Args: access_token: OAuth access token refresh_token: OAuth refresh token (for auto-refresh) expires_at: Unix timestamp when token expires config: MonzoConfig instance """ self.access_token = access_token self.refresh_token = refresh_token self.expires_at = expires_at or (time.time() + 21600) self.config = config or MonzoConfig self.base_url = self.config.BASE_URL def _get_headers(self) -> Dict[str, str]: """Generate request headers with auth token""" return { 'Authorization': f'Bearer {self.access_token}', 'Content-Type': 'application/x-www-form-urlencoded' } def _check_token_expiry(self): """Refresh token if expired (with 5-minute buffer)""" if self.expires_at and time.time() >= (self.expires_at - 300): if not self.refresh_token: raise ValueError("Token expired and no refresh_token available") # Refresh token token_data = refresh_access_token(self.refresh_token, self.config) self.access_token = token_data['access_token'] self.expires_at = time.time() + token_data['expires_in'] # Update refresh token if new one provided if 'refresh_token' in token_data: self.refresh_token = token_data['refresh_token'] def _request(self, method: str, endpoint: str, params: Optional[Dict] = None, data: Optional[Dict] = None) -> Dict: """ Make authenticated API request. Args: method: HTTP method (GET, POST, etc.) endpoint: API endpoint (e.g., '/accounts') params: URL query parameters data: Request body data Returns: JSON response as dict """ self._check_token_expiry() url = f"{self.base_url}{endpoint}" headers = self._get_headers() response = requests.request( method=method, url=url, headers=headers, params=params, data=data ) # Handle errors if response.status_code == 401: raise Exception("Unauthorized - token may be invalid") elif response.status_code == 429: raise Exception("Rate limit exceeded") response.raise_for_status() return response.json()
```

# Implementing Core API Methods

Add methods for the most common Monzo API operations:

## Get Accounts

```
def list_accounts(self) -> List[Dict]: """ List all accounts for authenticated user. Returns: List of account dictionaries """ response = self._request('GET', '/accounts') return response.get('accounts', []) def get_account_by_type(self, account_type: str = 'uk_retail') -> Optional[Dict]: """ Get account by type (uk_retail, uk_retail_joint). Args: account_type: Account type to filter by Returns: Account dict or None if not found """ accounts = self.list_accounts() for account in accounts: if account.get('type') == account_type and not account.get('closed'): return account return None
```

## Get Balance

```
def get_balance(self, account_id: str) -> Dict: """ Get current balance for account. Args: account_id: Account ID Returns: Balance dict with balance, total_balance, currency, spend_today """ params = {'account_id': account_id} return self._request('GET', '/balance', params=params)
```