perplexity

Open Geodata API - Complete User Guide

Table of Contents

Introduction

What is Open Geodata API?

Open Geodata API is a unified Python client library that provides seamless access to multiple open geospatial data APIs. It focuses on **API access, search, and URL management** while maintaining maximum flexibility for data reading and processing.

Key Features

✅ **Unified Access**: Single interface for multiple geospatial APIs
✅ **Automatic URL Management**: Handles signing (PC) and validation (ES) automatically
✅ **Maximum Flexibility**: Use any raster reading package you prefer
✅ **Zero Lock-in**: No forced dependencies or reading methods
✅ **Clean API**: Intuitive, Pythonic interface
✅ **Production Ready**: Robust error handling and comprehensive testing

Supported APIs

API	Provider	Authentication	URL Handling
Planetary Computer	Microsoft	API Key + Signing	Automatic signing
EarthSearch	Element84/AWS	None required	URL validation

Philosophy

 Core Focus: We provide URLs - you choose how to read them!
 Use Any Package: rioxarray, rasterio, GDAL, or any package you prefer
 Maximum Flexibility: Zero restrictions on your workflow

Installation

Basic Installation

```
# Install core package
pip install open-geodata-api
```

Optional Dependencies

```
# For spatial analysis (geopandas)
pip install open-geodata-api[spatial]

# For raster reading suggestions
pip install open-geodata-api[raster-rioxarray]  # rioxarray + xarray
pip install open-geodata-api[raster-rasterio]   # rasterio only
pip install open-geodata-api[raster-gdal]       # GDAL

# For plotting examples
pip install open-geodata-api[plotting]

# Development dependencies
pip install open-geodata-api[dev]
```

Verify Installation

```
import open_geodata_api as ogapi
ogapi.info()
```

Quick Start

30-Second Example

```
import open_geodata_api as ogapi

# Get clients for both APIs
clients = ogapi.get_clients(pc_auto_sign=True)
pc = clients['planetary_computer']
es = clients['earth_search']

# Search for Sentinel-2 data
results = pc.search(
    collections=["sentinel-2-l2a"],
    bbox=[-122.5, 47.5, -122.0, 48.0],
    datetime="2024-01-01/2024-03-31"
)

# Get items and URLs
items = results.get_all_items()
item = items[^0]

# Get ready-to-use URLs
blue_url = item.get_asset_url('B02')  # Automatically signed!
all_urls = item.get_all_asset_urls()  # All assets

# Use with ANY raster package
import rioxarray
data = rioxarray.open_rasterio(blue_url)

# Or use with rasterio
import rasterio
with rasterio.open(blue_url) as src:
    data = src.read(1)
```

5-Minute Tutorial

```
# 1. Import and setup
import open_geodata_api as ogapi

# 2. Create clients
pc = ogapi.planetary_computer(auto_sign=True)
es = ogapi.earth_search()

# 3. Search for data
search_params = {
    'collections': ['sentinel-2-l2a'],
    'bbox': [-122.5, 47.5, -122.0, 48.0],
    'datetime': '2024-01-01/2024-03-31',
    'query': {'eo:cloud_cover': {'lt': 30}}
}

pc_results = pc.search(**search_params, limit=10)
es_results = es.search(**search_params, limit=10)

# 4. Work with results
pc_items = pc_results.get_all_items()
es_items = es_results.get_all_items()

print(f"Found: PC={len(pc_items)}, ES={len(es_items)} items")

# 5. Get URLs and use with your preferred package
item = pc_items[^0]
item.print_assets_info()

# Get specific bands
rgb_urls = item.get_band_urls(['B04', 'B03', 'B02'])  # Red, Green, Blue
print(f"RGB URLs: {rgb_urls}")
```

```
# Use URLs with any package you want!
```

Core Concepts

STAC (SpatioTemporal Asset Catalog)

Open Geodata API works with STAC-compliant APIs. Key STAC concepts:

- **Collections**: Groups of related datasets (e.g., "sentinel-2-l2a")
- **Items**: Individual products/scenes with metadata
- **Assets**: Individual files (bands, thumbnails, metadata)

Package Architecture

```
open-geodata-api/
├── Core Classes (Universal)
│   ├── STACItem           # Individual products
│   ├── STACItemCollection # Groups of products
│   ├── STACAsset          # Individual files
│   └── STACSearch         # Search results
├── API Clients
│   ├── PlanetaryComputerCollections
│   └── EarthSearchCollections
└── Utilities
    ├── URL signing (PC)
    ├── URL validation (ES)
    └── Filtering functions
```

Provider-Specific Handling

Feature	Planetary Computer	EarthSearch
Authentication	Automatic via planetary-computer package	None required
URL Signing	Automatic (auto_sign=True)	Not applicable
Asset Naming	B01, B02, B03...	coastal, blue, green...
Cloud Cover	eo:cloud_cover	eo:cloud_cover

API Reference

Factory Functions

`planetary_computer(auto_sign=False)`

Creates a Planetary Computer client.

Parameters:

- `auto_sign` (bool): Automatically sign URLs for immediate use

Returns: `PlanetaryComputerCollections` instance

`earth_search(auto_validate=False)`

Creates an EarthSearch client.

Parameters:

- `auto_validate` (bool): Validate URLs (currently placeholder)

Returns: `EarthSearchCollections` instance

`get_clients(pc_auto_sign=False, es_auto_validate=False)`

Creates both clients simultaneously.

Returns: Dictionary with 'planetary_computer' and 'earth_search' keys

Client Methods

`search(collections, bbox=None, datetime=None, query=None, limit=100)`

Search for STAC items.

Parameters:

- `collections` (list): Collection IDs to search
- `bbox` (list): Bounding box [west, south, east, north]
- `datetime` (str): Date range "YYYY-MM-DD/YYYY-MM-DD"
- `query` (dict): Additional filters like `{"eo:cloud_cover": {"lt": 30}}`
- `limit` (int): Maximum results to return

Returns: `STACSearch` instance

`list_collections()`

Get list of available collection names.

Returns: List of collection ID strings `get_collection_info(collection_name)`

Get detailed information about a specific collection.

Returns: Collection metadata dictionary

STACItem Methods

`get_asset_url(asset_key, signed=None)`

Get ready-to-use URL for a specific asset.

Parameters:

- `asset_key` (str): Asset name (e.g., 'B02', 'blue', 'red')
- `signed` (bool): Override automatic signing behavior

Returns: URL string ready for any raster package

`get_all_asset_urls(signed=None)`

Get URLs for all available assets.

Returns: Dictionary `{asset_key: url}`

`get_band_urls(bands, signed=None)`

Get URLs for specific bands/assets.

Parameters:

- `bands` (list): List of asset names

Returns: Dictionary `{asset_key: url}`

`list_assets()`

Get list of available asset names.

Returns: List of asset key strings

`print_assets_info()`

Print detailed information about all assets.

STACItemCollection Methods

`get_all_urls(asset_keys=None, signed=None)`

Get URLs from all items in the collection.

Parameters:

- `asset_keys` (list, optional): Specific assets to get URLs for

- `signed` (bool, optional): Override signing behavior

Returns: Dictionary `{item_id: {asset_key: url}}`

`to_dataframe(include_geometry=True)`

Convert collection to pandas/geopandas DataFrame.

Parameters:

- `include_geometry` (bool): Include spatial geometry (requires geopandas)

Returns: DataFrame with item metadata

`export_urls_json(filename, asset_keys=None)`

Export all URLs to JSON file for external processing.

Usage Examples

Example 1: Simple Data Discovery

```
import open_geodata_api as ogapi

# Setup
pc = ogapi.planetary_computer(auto_sign=True)

# Find available collections
collections = pc.list_collections()
sentinel_collections = [c for c in collections if 'sentinel' in c.lower()]
print(f"Sentinel collections: {sentinel_collections}")

# Get collection details
s2_info = pc.get_collection_info('sentinel-2-l2a')
print(f"Sentinel-2 L2A: {s2_info['title']}")
print(f"Description: {s2_info['description'][:100]}...")
```

Example 2: Geographic Search

```
# Search around San Francisco Bay Area
bbox = [-122.5, 37.5, -122.0, 38.0]

results = pc.search(
    collections=['sentinel-2-l2a'],
    bbox=bbox,
    datetime='2024-06-01/2024-08-31',
    query={'eo:cloud_cover': {'lt': 20}},  # Less than 20% clouds
    limit=20
)

items = results.get_all_items()
print(f"Found {len(items)} items with <20% cloud cover")
```

```
# Convert to DataFrame for analysis
df = items.to_dataframe()
print(f"Date range: {df['datetime'].min()} to {df['datetime'].max()}")
print(f"Cloud cover range: {df['eo:cloud_cover'].min():.1f}% to {df['eo:cloud_cover'].max
```

Example 3: Multi-Provider Comparison

```
# Compare results from both providers
bbox = [-122.2, 47.6, -122.1, 47.7]  # Seattle area

pc_results = pc.search(
    collections=['sentinel-2-l2a'],
    bbox=bbox,
    datetime='2024-01-01/2024-03-31'
)

es_results = es.search(
    collections=['sentinel-2-l2a'],
    bbox=bbox,
    datetime='2024-01-01T00:00:00Z/2024-03-31T23:59:59Z'
)

pc_items = pc_results.get_all_items()
es_items = es_results.get_all_items()

print(f"Planetary Computer: {len(pc_items)} items")
print(f"EarthSearch: {len(es_items)} items")

# Compare asset availability
if pc_items and es_items:
    pc_assets = pc_items[^0].list_assets()
    es_assets = es_items[^0].list_assets()

    print(f"PC assets: {pc_assets[:5]}")
    print(f"ES assets: {es_assets[:5]}")
```

Example 4: URL Export for External Processing

```
# Get URLs for specific bands across multiple items
items = pc_results.get_all_items()

# Export RGB band URLs
rgb_urls = items.get_all_urls(['B04', 'B03', 'B02'])  # Red, Green, Blue

# Save to JSON for external processing
items.export_urls_json('sentinel2_rgb_urls.json', ['B04', 'B03', 'B02'])

# Use the URLs with any package
first_item_urls = rgb_urls[list(rgb_urls.keys())[^0]]
print(f"Red band URL: {first_item_urls['B04']}")

# Example with different raster packages
import rioxarray
import rasterio
from osgeo import gdal

red_url = first_item_urls['B04']

# Option 1: rioxarray
red_data_xr = rioxarray.open_rasterio(red_url)

# Option 2: rasterio
with rasterio.open(red_url) as src:
    red_data_rio = src.read(1)

# Option 3: GDAL
red_ds = gdal.Open(red_url)
red_data_gdal = red_ds.ReadAsArray()

print(f"Data shapes - XR: {red_data_xr.shape}, RIO: {red_data_rio.shape}, GDAL: {red_data
```

Example 5: Batch Processing Setup

```
# Setup for batch processing
import json

# Search for monthly data
results = pc.search(
    collections=['sentinel-2-l2a'],
    bbox=[-120.0, 35.0, -119.0, 36.0],
    datetime='2024-01-01/2024-12-31',
    query={'eo:cloud_cover': {'lt': 15}},
    limit=100
)

items = results.get_all_items()
print(f"Found {len(items)} low-cloud scenes")

# Group by month
df = items.to_dataframe()
df['month'] = df['datetime'].str[:7]  # YYYY-MM
monthly_counts = df.groupby('month').size()
print("Monthly data availability:")
print(monthly_counts)

# Export all URLs for batch processing
all_urls = items.get_all_urls(['B04', 'B03', 'B02', 'B08'])  # RGB + NIR

# Save configuration for external processing
config = {
    'search_params': {
        'bbox': [-120.0, 35.0, -119.0, 36.0],
        'datetime': '2024-01-01/2024-12-31',
        'collections': ['sentinel-2-l2a']
    },
    'items_found': len(items),
    'urls': all_urls
}
```

```
with open('batch_processing_config.json', 'w') as f:
    json.dump(config, f, indent=2)

print("Batch processing configuration saved!")
```

Example 6: EarthSearch Specific Features

```
# EarthSearch uses different asset names
es = ogapi.earth_search()

es_results = es.search(
    collections=['sentinel-2-l2a'],
    bbox=[-122.5, 47.5, -122.0, 48.0],
    datetime='2024-06-01T00:00:00Z/2024-08-31T23:59:59Z',
    limit=5
)

es_items = es_results.get_all_items()
item = es_items[^0]

# EarthSearch asset names
item.print_assets_info()

# Get URLs using EarthSearch naming
rgb_urls = item.get_band_urls(['red', 'green', 'blue'])
nir_url = item.get_asset_url('nir')

print(f"RGB URLs: {list(rgb_urls.keys())}")
print(f"NIR URL ready: {nir_url[:50]}...")

# All URLs (no signing needed for EarthSearch)
all_urls = item.get_all_asset_urls()
print(f"Total assets available: {len(all_urls)}")
```

Best Practices

1. Client Configuration

```
# Recommended setup
import open_geodata_api as ogapi

# Auto-sign PC URLs for immediate use
pc = ogapi.planetary_computer(auto_sign=True)
es = ogapi.earth_search()

# Or get both at once
clients = ogapi.get_clients(pc_auto_sign=True)
```

2. Search Strategy

```
# Start with broad search, then refine
results = pc.search(
    collections=['sentinel-2-l2a'],
    bbox=your_bbox,
    datetime='2024-01-01/2024-12-31',
    query={'eo:cloud_cover': {'lt': 50}},  # Start broad
    limit=100
)

# Filter further based on your needs
df = results.get_all_items().to_dataframe()
filtered_df = df[df['eo:cloud_cover'] < 20]  # Refine cloud cover
```

3. URL Management

```
# Let the package handle URL signing automatically
item = items[^0]

# This automatically handles signing based on provider
blue_url = item.get_asset_url('B02')  # PC: signed, ES: validated

# Override if needed
unsigned_url = item.get_asset_url('B02', signed=False)
```

4. Asset Name Handling

```
# Handle different naming conventions gracefully
def get_rgb_urls(item):
    """Get RGB URLs regardless of provider naming."""
    assets = item.list_assets()

    # Try Planetary Computer naming
    if all(band in assets for band in ['B04', 'B03', 'B02']):
        return item.get_band_urls(['B04', 'B03', 'B02'])

    # Try EarthSearch naming
    elif all(band in assets for band in ['red', 'green', 'blue']):
        return item.get_band_urls(['red', 'green', 'blue'])

    else:
        print(f"Available assets: {assets}")
        return {}

# Use the function
rgb_urls = get_rgb_urls(item)
```

5. Error Handling

```
# Robust search with error handling
def safe_search(client, **kwargs):
    """Search with comprehensive error handling."""
    try:
        results = client.search(**kwargs)
        items = results.get_all_items()

        if len(items) == 0:
            print("No items found. Try adjusting search parameters.")
            return None

        print(f"Found {len(items)} items")
        return items

    except Exception as e:
        print(f"Search failed: {e}")
        return None

# Use robust search
items = safe_search(
    pc,
    collections=['sentinel-2-l2a'],
    bbox=your_bbox,
    datetime='2024-01-01/2024-03-31'
)
```

6. Memory Management

```
# For large datasets, process in batches
def process_in_batches(items, batch_size=10):
    """Process items in batches to manage memory."""
    for i in range(0, len(items), batch_size):
        batch = items[i:i+batch_size]

        # Get URLs for this batch
        batch_urls = {}
        for item in batch:
            try:
                batch_urls[item.id] = item.get_band_urls(['B04', 'B03', 'B02'])
            except Exception as e:
                print(f"Failed to get URLs for {item.id}: {e}")

        # Process batch_urls as needed
        yield batch_urls

# Use batch processing
for batch_urls in process_in_batches(items):
    print(f"Processing batch with {len(batch_urls)} items")
    # Your processing logic here
```

Troubleshooting

Common Issues and Solutions

Issue: "planetary-computer package not found"

Problem: PC URL signing fails

```
# Error: planetary-computer package not found, returning unsigned URL
```

Solution:

```
pip install planetary-computer
```

Issue: No items found

Problem: Search returns empty results

Solutions:

```
# 1. Check collection names
available_collections = pc.list_collections()
print("Available collections:", available_collections)

# 2. Expand search area
bbox = [-123.0, 47.0, -121.0, 48.0]  # Larger area

# 3. Expand date range
datetime = '2023-01-01/2024-12-31'  # Larger time window

# 4. Relax cloud cover
query = {'eo:cloud_cover': {'lt': 80}}  # More permissive
```

Issue: Asset not found

Problem: `KeyError: Asset 'B02' not found`

Solutions:

```
# 1. Check available assets
item.print_assets_info()

# 2. Use correct naming for provider
# PC: B01, B02, B03...
# ES: coastal, blue, green...

# 3. Handle gracefully
try:
    url = item.get_asset_url('B02')
```

```
except KeyError:
    # Try alternative naming
    url = item.get_asset_url('blue')
```

Issue: EarthSearch datetime format

Problem: EarthSearch requires RFC3339 format

Solution:

```
# Use proper format for EarthSearch
datetime_es = '2024-01-01T00:00:00Z/2024-03-31T23:59:59Z'

# Package handles this automatically in most cases
```

Issue: Large data downloads

Problem: Memory issues with large datasets

Solutions:

```
# 1. Use overview levels (if your raster package supports it)
import rioxarray
data = rioxarray.open_rasterio(url, overview_level=2)

# 2. Use chunking
data = rioxarray.open_rasterio(url, chunks={'x': 512, 'y': 512})

# 3. Read windows
import rasterio
with rasterio.open(url) as src:
    window = rasterio.windows.Window(0, 0, 1024, 1024)
    data = src.read(1, window=window)
```

Debug Mode

```
# Enable debug information
import logging
logging.basicConfig(level=logging.DEBUG)

# Check what URLs are being generated
item = items[^0]
print(f"Item ID: {item.id}")
print(f"Provider: {item.provider}")

all_urls = item.get_all_asset_urls()
for asset, url in all_urls.items():
    print(f"{asset}: {url[:50]}...")
```

Validation Steps

```
# Validate your setup
def validate_setup():
    """Validate package installation and API access."""
    try:
        import open_geodata_api as ogapi
        print("✅ Package imported successfully")

        # Test client creation
        pc = ogapi.planetary_computer()
        es = ogapi.earth_search()
        print("✅ Clients created successfully")

        # Test collection listing
        pc_collections = pc.list_collections()
        print(f"✅ PC collections: {len(pc_collections)} available")

        # Test simple search
        test_results = pc.search(
            collections=['sentinel-2-l2a'],
            bbox=[-122.0, 47.0, -121.0, 48.0],
            limit=1
        )
        test_items = test_results.get_all_items()
        print(f"✅ Test search: {len(test_items)} items found")

        return True

    except Exception as e:
        print(f"❌ Validation failed: {e}")
        return False

# Run validation
validate_setup()
```

Advanced Usage

Custom Processing Workflows

```
# Example: Multi-temporal analysis setup
def setup_temporal_analysis(bbox, date_ranges, max_cloud_cover=20):
    """Setup data for temporal analysis."""

    all_data = {}

    for period_name, date_range in date_ranges.items():
        print(f"Searching for {period_name}...")

        results = pc.search(
            collections=['sentinel-2-l2a'],
            bbox=bbox,
            datetime=date_range,
            query={'eo:cloud_cover': {'lt': max_cloud_cover}},
            limit=50
        )

        items = results.get_all_items()
        urls = items.get_all_urls(['B04', 'B03', 'B02', 'B08'])  # RGB + NIR

        all_data[period_name] = {
            'count': len(items),
            'date_range': date_range,
            'urls': urls
        }

        print(f"  Found {len(items)} items")

    return all_data

# Use for seasonal analysis
seasonal_data = setup_temporal_analysis(
    bbox=[-120.0, 35.0, -119.0, 36.0],
    date_ranges={
        'spring_2024': '2024-03-01/2024-05-31',
        'summer_2024': '2024-06-01/2024-08-31',
        'fall_2024': '2024-09-01/2024-11-30'
    }
)
```

Integration with Other Libraries

```
# Example: Integration with STAC-tools
def integrate_with_stac_tools(items):
    """Convert to format compatible with other STAC tools."""

    # Export as standard STAC format
    stac_collection = items.to_dict()  # GeoJSON FeatureCollection

    # Use with pystac
    try:
        import pystac

        # Convert items for pystac
        pystac_items = []
        for item_data in items.to_list():
            pystac_item = pystac.Item.from_dict(item_data)
            pystac_items.append(pystac_item)

        print(f"Converted {len(pystac_items)} items to pystac format")
        return pystac_items

    except ImportError:
        print("pystac not available")
        return stac_collection

# Example: Integration with stackstac
def prepare_for_stackstac(items, bands=['B04', 'B03', 'B02']):
    """Prepare data for stackstac processing."""

    try:
        import stackstac

        # Get STAC items in proper format
        stac_items = [item.to_dict() for item in items]

        # Note: URLs need to be properly signed
        # The package handles this automatically

        print(f"Prepared {len(stac_items)} items for stackstac")
        print(f"Bands: {bands}")

        return stac_items

    except ImportError:
        print("stackstac not available")
        return None
```

Custom URL Processing

```
# Example: Custom URL validation and processing
def process_urls_custom(items, custom_processor=None):
    """Process URLs with custom logic."""

    def default_processor(url):
        """Default URL processor."""
        # Add custom headers, caching, etc.
        return url

    processor = custom_processor or default_processor

    processed_urls = {}

    for item in items:
        item_urls = item.get_all_asset_urls()
        processed_item_urls = {}

        for asset, url in item_urls.items():
            processed_url = processor(url)
            processed_item_urls[asset] = processed_url

        processed_urls[item.id] = processed_item_urls

    return processed_urls

# Example custom processor
def add_caching_headers(url):
    """Add caching parameters to URL."""
    if '?' in url:
        return f"{url}&cache=3600"
    else:
        return f"{url}?cache=3600"
```

```
# Use custom processing
cached_urls = process_urls_custom(items, add_caching_headers)
```

FAQ

General Questions

Q: What makes this package different from using APIs directly?

A: Key advantages:

- Unified interface across multiple APIs
- Automatic URL signing/validation
- Consistent error handling
- No lock-in to specific data reading packages
- Built-in best practices

Q: Can I use this with my existing geospatial workflow?

A: Absolutely! The package provides URLs that work with any raster reading library:

```
url = item.get_asset_url('red')

# Use with your existing tools
import rioxarray; data = rioxarray.open_rasterio(url)
import rasterio; data = rasterio.open(url)
from osgeo import gdal; data = gdal.Open(url)
```

Q: Do I need API keys?

A: Only for Planetary Computer. EarthSearch is completely open.

Technical Questions

Q: How does automatic URL signing work?

A: When `auto_sign=True`, the package:

1. Detects the provider (PC vs ES)
2. For PC: Uses the planetary-computer package to sign URLs
3. For ES: Returns URLs as-is (no signing needed)
4. You can override with `signed=False/True`

Q: What about rate limiting?

A: Both APIs have rate limits:

- **Planetary Computer**: Generous limits for signed URLs

- **EarthSearch**: Standard HTTP rate limits

The package doesn't implement rate limiting - use your own if needed.

Q: Can I cache results?

A: Yes, several approaches:

```
# 1. Export URLs to JSON
items.export_urls_json('cache.json')

# 2. Save DataFrames
df = items.to_dataframe()
df.to_parquet('metadata_cache.parquet')

# 3. Use your own caching layer
```

Q: How do I handle different projections?

A: The package provides URLs - projection handling is up to your raster library:

```
import rioxarray
data = rioxarray.open_rasterio(url)
data_reprojected = data.rio.reproject('EPSG:4326')
```

Troubleshooting Questions

Q: Why am I getting "Asset not found" errors?

A: Different providers use different asset names:

- **PC**: B01, B02, B03, B04...
- **EarthSearch**: coastal, blue, green, red...

Use `item.print_assets_info()` to see available assets.

Q: Search returns no results but data should exist

A: Common issues:

1. **Bbox order**: Use [west, south, east, north]
2. **Date format**: PC accepts "YYYY-MM-DD", ES prefers RFC3339
3. **Collection names**: Use `client.list_collections()` to verify
4. **Cloud cover**: Try relaxing the threshold

Q: URLs work but data loading is slow

A: Optimization strategies:

1. Use overview levels: `rioxarray.open_rasterio(url, overview_level=2)`

2. Enable chunking: `rioxarray.open_rasterio(url, chunks=True)`
3. Read smaller windows with rasterio
4. Consider geographic proximity to data

Integration Questions

Q: Can I use this with Jupyter notebooks?

A: Yes! The package works great in Jupyter:

```
# Display asset info
item.print_assets_info()

# Show DataFrames
df = items.to_dataframe()
display(df)

# Plot with matplotlib/cartopy
import matplotlib.pyplot as plt
data = rioxarray.open_rasterio(url)
data.plot()
```

Q: How do I integrate with QGIS/ArcGIS?

A: Export URLs and use them directly:

```
# Get URLs
urls = item.get_all_asset_urls()

# In QGIS: Add Raster Layer -> use the URL directly
# In ArcGIS: Add Data -> Raster Dataset -> paste URL
```

Q: Can I use this in production systems?

A: Yes! The package is designed for production use:

- Robust error handling
- No forced dependencies
- Clean separation of concerns
- Comprehensive logging support

Q: How do I contribute or report issues?

A: Visit the GitHub repository:

- Report issues: GitHub Issues
- Contribute: Pull Requests welcome
- Documentation: Help improve this guide

This completes the comprehensive user guide for Open Geodata API. The package provides a clean, flexible foundation for accessing open geospatial data while letting you maintain full control over data processing and analysis workflows.

⁂