Swiftshadow

🔗

PyPI - Downloads

GitHub release (latest by date including pre-releases)

⚠Warning

Heads up! If you're using **versions 1.2.1 or below**, please note that **version 2.0.0 and above** includes breaking changes. Before upgrading, read the documentation to understand the changes and ensure compatibility with your code. If you encounter issues, please review the docs before opening a GitHub issue.

About

🔗

Swiftshadow is a lightweight and efficient Python library designed to simplify IP proxy rotation for web scraping, data mining, and other automated tasks. It helps you bypass common challenges like IP bans, rate limits, and detection mechanisms, ensuring smooth and uninterrupted data collection.

Key Features

- **Speed**: Optimized for fast proxy retrieval and rotation.
- **Reliability**: Automatically switches to working proxies if one fails.
- **Customization**: Configure proxy filters, rotation frequency, and caching behavior.
- **Low Dependencies**: Only one third-party dependency (`requests`), making it easy to use and maintain.
- **Caching**: Built-in caching mechanism to reduce load times and improve performance.

Whether you're a seasoned developer or a beginner, **Swiftshadow** makes proxy management effortless.

Installation

Install the library using pip:

```
pip install swiftshadow
```

Quick Start

Get a Proxy in 2 Lines

Fetch a random proxy with just two lines of code:

```
from swiftshadow import QuickProxy

print(QuickProxy())  # Output: http://<ip>:<port>
```

Advanced Usage

For more control, use the `ProxyInterface` class:

```
from swiftshadow.classes import ProxyInterface

# Fetch HTTPS proxies from the US
swift = ProxyInterface(countries=["US"], protocol="https")
print(swift.get().as_string())  # Output: https://<ip>:<port>
```

Documentation

🔗

Explore the full documentation to learn more about **Swiftshadow**'s features and advanced usage:

📚 Documentation

Why Swiftshadow?

🔗

- **Lightweight**: Minimal dependencies and easy to integrate.
- **Flexible**: Supports filtering by country and protocol.
- **Scalable**: Ideal for both small scripts and large-scale scraping projects.
- **Open Source**: Free to use, modify, and contribute to under the MIT License.

Contributing

🔗

Contributions are welcome! If you'd like to improve **Swiftshadow**, feel free to open an issue or submit a pull request on GitHub.

License

Swiftshadow is licensed under the MIT License. See the LICENSE file for details.

Providers

Swiftshadow is a service that provides a range of proxies to users who require online privacy and security. These proxies are sourced from free proxy lists.

I want to take a moment to express gratitude to the providers mentioned below for generously supplying us with free proxy lists.

1. **Monosans**
   GitHub: monosans/proxy-list

2. **TheSpeedX**
   GitHub: TheSpeedX/PROXY-List

3. **GoodProxy**
   GitHub: yuceltoluyag/GoodProxy

4. **MuRongPIG**
   GitHub: MuRongPIG/Proxy-Master

5. **KangProxy**
   GitHub: officialputuid/KangProxy

6. **Mmpx12**
   GitHub: mmpx12/proxy-list

7. **Anonym0usWork1221**
   GitHub: Anonym0usWork1221/Free-Proxies

8. **ProxySpace**
   GitHub: proxyspace/proxyspace

9. **ProxyScrape**
10. **OpenProxyList**

ProxyInterface Class

`swiftshadow.classes.ProxyInterface`

Manages proxy acquisition, caching, and rotation from various providers.

This class handles proxy retrieval either through fresh fetching from registered providers or via cached data. It supports protocol filtering, country filtering, cache management, and automatic/manual proxy rotation.

Attributes:

Name	Type	Description
`countries`	`list[str]`	List of ISO country codes to filter proxies by (e.g., ["US", "CA"]).
`protocol`	`Literal['https', 'http']`	Proxy protocol to use. Defaults to 'http'.
`maxproxies`	`int`	Maximum number of proxies to collect from providers. Defaults to 10.
`autorotate`	`bool`	Whether to automatically rotate proxy on each get()

Name	Type	Description
		call. Defaults to False.
`autoUpdate`	`bool`	Whether to automatically update proxies upon class initalisation. Defaults to True.
`cachePeriod`	`int`	Number of minutes before cache is considered expired. Defaults to 10.
`cacheFolderPath`	`Path`	Filesystem path for cache storage. Uses system cache dir by default.
`proxies`	`list[Proxy]`	List of available proxy objects.
`current`	`Proxy | None`	Currently active proxy. None if no proxies available.
`cacheExpiry`	`datetime | None`	Timestamp when cache expires. None if no cache exists.

▶ Example

Raises:

Type	Description
`UnsupportedProxyProtocol`	If invalid protocol is specified during initialization.
`ValueError`	If no proxies match filters during update().

▶ Source code in `swiftshadow/classes.py`

`__init__(countries=[], protocol='http', maxProxies=10, autoRotate=False, autoUpdate=True, cachePeriod=10, cacheFolderPath=None, debug=False, logToFile=False)`

Initializes ProxyInterface with specified configuration.

Parameters:

Name	Type	Description	Default
`countries`	`list[str]`	List of ISO country codes to filter proxies. Empty list = no filtering.	`[]`
`protocol`	`Literal['https', 'http']`	Proxy protocol to retrieve. Choose between 'http' or 'https'.	`'http'`

Name	Type	Description	Default
`maxProxies`	`int`	Maximum proxies to collect from all providers combined.	`10`
`autoRotate`	`bool`	Enable automatic proxy rotation on every get() call.	`False`
`autoUpdate`	`bool`	Whether to automatically update proxies upon class initalisation.	`True`
`cachePeriod`	`int`	Cache validity duration in minutes.	`10`
`cacheFolderPath`	`Path | None`	Custom path for cache storage. Uses system cache dir if None.	`None`
`debug`	`bool`	Enable debug logging level when True.	`False`
`logToFile`	`bool`	Write logs to swiftshadow.log in cache folder when True.	`False`

▶ Source code in `swiftshadow/classes.py`

`async_update() async`

Updates proxy list from providers or cache in async.

First attempts to load valid proxies from cache. If cache is expired/missing, fetches fresh proxies from registered providers that match country and protocol filters. Updates cache file with new proxies if fetched from providers.

Raises:

Type	Description
`ValueError`	If no proxies found after provider scraping.

▶ Source code in `swiftshadow/classes.py`

`get()`

Retrieves current active proxy.

Returns:

Name	Type	Description
`Proxy`	`Proxy`	Current proxy object with connection details.

▶ Note

Raises:

Type	Description
`ValueError`	If no proxies are available (current is None).

▶ Source code in `swiftshadow/classes.py`

`rotate(validate_cache=False)`

Rotates to a random proxy from available proxies.

Parameters:

Name	Type	Description	Default
`validate_cache`	`bool`	Force cache validation before rotation when True.	`False`

▶ Note

Raises:

Type	Description
`ValueError`	If validate_cache=True but no cache exists.

▶ Source code in `swiftshadow/classes.py`

`update()`

Updates proxy list from providers or cache.

First attempts to load valid proxies from cache. If cache is expired/missing, fetches fresh proxies from registered providers that match country and protocol filters. Updates cache file with new proxies if fetched from providers.

Raises:

Type	Description
`ValueError`	If no proxies found after provider scraping.

▶ Source code in `swiftshadow/classes.py`

QuickProxy Class

`swiftshadow.QuickProxy(countries=[], protocol='http')`

This function is a faster alternative to `ProxyInterface` class. No caching is done.

Parameters:

Name	Type	Description	Default
`countries`	`list[str]`	ISO 3166-2 Two letter country codes to filter proxies.	`[]`
`protocol`	`Literal['http', 'https']`	HTTP/HTTPS protocol to filter proxies.	`'http'`

Returns:

Name	Type	Description
`proxyObject`	`Proxy`	A working proxy object if found or else None.

▶ Source code in `swiftshadow/__init__.py`

CacheData Model

`swiftshadow.models.CacheData` `dataclass`

Class repersenting data structure if the cache in cache file.

Attributes:

Name	Type	Description
`expiryIn`	`datetime`	Expiry date object.
`proxies`	`list[Proxy]`	Proxies to head.

▶ Source code in `swiftshadow/models.py`

Provider Model

`swiftshadow.models.Provider` dataclass

Class repersenting a Provider.

Attributes:

Name	Type	Description
`providerFunction`	`Callable[[list[str], Literal['http', 'https']], Coroutine[Any, Any, list[]]]`	Callable function for this provider.
`countryFilter`	`bool`	Whether the provider supports country based filters.
`protocols`	`list[Literal['http', 'https']]`	Protocols supported by the provider.

▶ Source code in `swiftshadow/models.py`

Proxy Model

`swiftshadow.models.Proxy` dataclass

Class representing a Proxy object.

Attributes:

Name	Type	Description
`ip`	`str`	IP Address of the proxy.
`port`	`int`	Port associated with the proxy.
`protocol`	`Literal['http', 'https']`	Protocol type of the proxy.

▶ Source code in `swiftshadow/models.py`

`as_requests_dict()`

Return proxy in requests commpatible dict format.

Returns:

Name	Type	Description
`dict`	`dict[Literal['http', 'https'], str]`	Dict representation of Proxy class.

▶ Source code in `swiftshadow/models.py`

`as_string()`

Return proxy in a string of format ://:

Returns:

Name	Type	Description
`string`	`str`	Proxy in string format.

▶ Source code in `swiftshadow/models.py`

Usage in Async Environments

SwiftShadow's default `update` function uses `asyncio.run()` to run its provider coroutines. This can lead to issues when using async frameworks like **FastAPI** or **Quart**, which already manage their own event loops.

Avoid `asyncio.run()` in Async Apps

Running `asyncio.run()` inside an existing event loop (like those in FastAPI or Quart) may cause errors or unexpected behavior.

The Simple Solution

Instead of using `update`, call `async_update()` to refresh your proxies. This makes sure your proxies are up to date and safe to use in your async app.

Keep Proxies Fresh

Create a background task that periodically calls `async_update()` so your proxies are always valid.

FastAPI Example

Use this example in FastAPI to update proxies every 5 seconds:

```python
from fastapi import FastAPI
import asyncio
from swiftshadow.classes import ProxyInterface

app = FastAPI()

# Create the ProxyInterface with autoUpdate disabled.
swift = ProxyInterface(autoUpdate=False, autoRotate=True)

async def background_task():
    """Update proxies every 5 seconds."""
    while True:
        print("Updating proxies...")
        await swift.async_update()
        await asyncio.sleep(5)

@app.on_event("startup")
async def startup_event():
    """Start the background task when the app starts."""
    asyncio.create_task(background_task())
    print("Background task started!")

@app.get("/")
async def home():
    """Return a refreshed proxy."""
    proxy = swift.get()
    return {
        "message": "Hello, FastAPI! Here is a proxy.",
        "proxy": proxy.as_string()
    }

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="127.0.0.1", port=8000, debug=True)
```

Quart Example

Below is a similar example for Quart. In this updated example, we disable automatic rotation (i.e. `autoRotate=False`) and instead call `rotate(validate_cache=False)` manually within the route. This avoids errors related to the event loop in Quart.

```
from quart import Quart
import asyncio
from swiftshadow.classes import ProxyInterface

app = Quart(__name__)
swift = ProxyInterface(autoUpdate=False, autoRotate=False)  #
manual update and rotation

async def background_task():
    """Update proxies every 5 seconds."""
    while True:
        print("Updating proxies...")
        await swift.async_update()
        await asyncio.sleep(5)

@app.before_serving
async def startup():
    """Start the background task when the server starts."""
    app.add_background_task(background_task)
    print("Background task started!")

@app.route("/")
async def home():
    """Return a refreshed proxy."""
    swift.rotate(validate_cache=False)  # manually rotate
without cache validation
    return "Hello, Quart! Here is a proxy: " +
swift.get().as_string()

if __name__ == "__main__":
    app.run(debug=True)
```

Summary

By calling `async_update()` in a background task, you ensure that your proxies are refreshed safely within your app's own event loop.
Note: In some async frameworks (e.g., Quart), if you encounter issues with auto-rotation, consider disabling `autoRotate` and manually calling `rotate(validate_cache=False)` to avoid event loop conflicts.

Remember

Always call `async_update()` before accessing a proxy to keep it up to date and avoid potential issues with event loops.

Basic Usage

To get a random `HTTP` proxy from any country:

```
from swiftshadow.classes import ProxyInterface
swift = ProxyInterface()

print(swift.get().as_string())
```

Note

When the `ProxyInterface` class instance is created for the first time, the `update()` method is called. This method fetches proxies from providers and caches them. This process may take around 10 seconds. The cache will refresh after the `cachePeriod` expires.

The `get()` method returns a `Proxy` object. You can convert it to a string using `as_string()` or to a dictionary using `as_requests_dict()`. See the References for more details.

This is the most basic usage of `swiftshadow`, but there's more to explore.

Note

From now on, all examples will exclude the import statement for simplicity.

Filtering Proxies

You can filter proxies based on their country of origin or protocol (HTTP/HTTPS).

Country Filter

To filter proxies by country, pass a list of **two-letter country codes** when initializing the `ProxyInterface` class.

- For a list of countries and their two-letter codes, visit this Wikipedia page.

Country Filtered

```
swift = ProxyInterface(countries=['US', 'IN'])
```

Protocol Filter

By default, all proxies are `HTTP`. For **SSL-enabled** `HTTPS` proxies, set the `protocol` parameter to `"https"`.

HTTPS Filter

```
swift = ProxyInterface(protocol='https')
```

Warning

`swiftshadow` does not validate country codes or protocols. If you provide invalid country codes or protocols, no proxies will be available.

Proxy Rotation

Manual Rotation

You can manually rotate proxies using the `rotate()` method. This selects a random proxy from the available list.

```
from swiftshadow.classes import ProxyInterface
swift = ProxyInterface()

print(swift.get().as_string())
swift.rotate()
print(swift.get().as_string())
```

Auto Rotation

To enable automatic proxy rotation, set the `autoRotate` parameter to `True` when initializing the `ProxyInterface` object. When enabled, the proxy will rotate every time the `get()` method is called.

```
from swiftshadow.classes import ProxyInterface
swift = ProxyInterface(autoRotate=True)

print(swift.get().as_string())
print(swift.get().as_string())
```

Caching

Proxies are cached to improve performance. The cache expires after the `cachePeriod` (default: 10 minutes). You can force a cache update by calling the `update()` method.

```
swift = ProxyInterface()
swift.update()  # Force update the proxy list
```

Visit References for more information on methods and classes.

QuickProxy

For faster use cases where caching is not required, the `swiftshadow.QuickProxy` function is the best choice. Unlike the `ProxyInterface` class, `QuickProxy` does not cache proxies, making it lightweight and ideal for one-off or quick operations.

`swiftshadow.QuickProxy(countries=[], protocol='http')`

This function is a faster alternative to `ProxyInterface` class. No caching is done.

Parameters:

Name	Type	Description	Default
`countries`	`list[str]`	ISO 3166-2 Two letter country codes to filter proxies.	`[]`
`protocol`	`Literal['http', 'https']`	HTTP/HTTPS protocol to filter proxies.	`'http'`

Returns:

Name	Type	Description
`proxyObject`	`Proxy`	A working proxy object if found or else None.

▶ Source code in `swiftshadow/__init__.py`

You can use filters just like in the `ProxyInterface` class. This includes filtering by **country** and **protocol**.

Parameters

- `countries` (list[str]): A list of two-letter country codes to filter proxies by country. Defaults to an empty list (no filtering).
- `protocol` (Literal["http", "https"]): The protocol to filter proxies by. Defaults to `"http"`.

Returns

- `Proxy` | `None`: A `Proxy` object if a valid proxy is found, otherwise `None`.

Example

```
from swiftshadow import QuickProxy

# Get a random HTTP proxy
proxy = QuickProxy()
```

```
print(proxy.as_string())

# Get an HTTPS proxy from the US or India
proxy = QuickProxy(countries=["US", "IN"], protocol="https")
print(proxy.as_string())
```

Note

Since `QuickProxy` does not cache proxies, it may take slightly longer to fetch a proxy compared to `ProxyInterface` when used repeatedly. However, it is faster for single-use scenarios.

Warning

If no proxies match the provided filters, `QuickProxy` will return `None`. Always check the return value before using it.

For more advanced use cases, such as caching and automatic rotation, consider using the `ProxyInterface` class.

Examples

This page provides practical examples to help you get started with `swiftshadow`. It covers all major features, including filtering, caching, rotation, and integration with popular libraries.

Basic Usage

Fetching a Proxy

To fetch a random proxy:

```
from swiftshadow import QuickProxy

proxy = QuickProxy()
print(proxy.as_string())  # Output: http://<ip>:<port>
```

Filtering Proxies

Country Filter

Filter proxies by country using two-letter ISO codes:

```
from swiftshadow.classes import ProxyInterface

# Fetch proxies from the US and India
swift = ProxyInterface(countries=["US", "IN"])
print(swift.get().as_string())
```

Protocol Filter

Filter proxies by protocol (`http` or `https`):

```
from swiftshadow.classes import ProxyInterface

# Fetch HTTPS proxies
swift = ProxyInterface(protocol="https")
print(swift.get().as_string())
```

Proxy Rotation

Manual Rotation

Manually rotate proxies using the `rotate()` method:

```
from swiftshadow.classes import ProxyInterface

swift = ProxyInterface()
```

```
# Get the first proxy
print(swift.get().as_string())

# Rotate to a new proxy
swift.rotate()
print(swift.get().as_string())
```

Automatic Rotation

Enable automatic rotation by setting `autoRotate=True`:

```
from swiftshadow.classes import ProxyInterface

swift = ProxyInterface(autoRotate=True)

# Each call to get() will return a new proxy
print(swift.get().as_string())
print(swift.get().as_string())
```

Caching

Custom Cache Folder

Specify a custom cache folder (useful for AWS Lambda):

```
from swiftshadow.classes import ProxyInterface

# Use the /tmp directory for caching
```

```
swift = ProxyInterface(cacheFolderPath="/tmp")
print(swift.get().as_string())
```

Disabling Caching

For one-off use cases, use `QuickProxy`:

```
from swiftshadow import QuickProxy

proxy = QuickProxy()
print(proxy.as_string())
```

Integration with Libraries

Using with **`requests`**

Route requests through a proxy using the `requests` library:

```
from swiftshadow import QuickProxy
from requests import get

proxy = QuickProxy()
resp = get('https://checkip.amazonaws.com',
proxies=proxy.as_requests_dict())
print(resp.text)  # Output: Proxy's IP address
```

Using with **`httpx`**

Route requests through a proxy using the `httpx` library:

```
from swiftshadow import QuickProxy
import httpx

proxy = QuickProxy()
with httpx.Client(proxies={"http://": proxy.as_string(),
"https://": proxy.as_string()}) as client:
    resp = client.get('https://checkip.amazonaws.com')
    print(resp.text)  # Output: Proxy's IP address
```

Advanced Usage

Combining Filters

Combine country and protocol filters for precise proxy selection:

```
from swiftshadow.classes import ProxyInterface

# Fetch HTTPS proxies from the US and India
swift = ProxyInterface(countries=["US", "IN"],
protocol="https")
print(swift.get().as_string())
```

Force Cache Update

Force an update of the proxy cache:

```
from swiftshadow.classes import ProxyInterface

swift = ProxyInterface()
swift.update()  # Force update the proxy list
print(swift.get().as_string())
```

For more details on classes and methods, visit the References page.

AWS Lambda

Using `swiftshadow` with AWS Lambda normally raises an error because the cache mechanism won't work properly due to the read-only file permissions in the Lambda environment. To fix this, you can pass the `cacheFolderPath` parameter and set it to `"/tmp"` when creating a `ProxyInterface` instance.

```
from swiftshadow.classes import ProxyInterface

swift = ProxyInterface(cacheFolderPath="/tmp")
```

The `/tmp` directory in AWS Lambda is writable, allowing the cache to function correctly.

Disabling Caching

If you don’t want the caching behavior at all, consider using the `QuickProxy` function instead. It does not cache proxies, making it ideal for serverless environments like AWS Lambda.

```
from swiftshadow import QuickProxy

proxy = QuickProxy()
print(proxy.as_string())
```

For more details on the `ProxyInterface` class or the `QuickProxy` function, visit the References page.

HTTPX Library

`swiftshadow` works seamlessly with the modern and asynchronous `httpx` library, making it easy to route your HTTP/HTTPS requests through a proxy. Whether you're building synchronous or asynchronous applications, `swiftshadow` has you covered.

Example Usage

Here’s how you can use a proxy fetched by `QuickProxy` with the `httpx` library:

Synchronous Example

```
from swiftshadow import QuickProxy
import httpx

# Fetch a proxy
proxy = QuickProxy()

# Use the proxy with httpx
with httpx.Client(proxies={"http://": proxy.as_string(),
"https://": proxy.as_string()}) as client:
    resp = client.get('https://checkip.amazonaws.com')
    print(resp.text)
```

Asynchronous Example

```
from swiftshadow import QuickProxy
import httpx
import asyncio

async def fetch_with_proxy():
    # Fetch a proxy
    proxy = QuickProxy()

    # Use the proxy with httpx
    async with httpx.AsyncClient(proxies={"http://":
proxy.as_string(), "https://": proxy.as_string()}) as client:
        resp = await
client.get('https://checkip.amazonaws.com')
        print(resp.text)

# Run the async function
asyncio.run(fetch_with_proxy())
```

Explanation

- `QuickProxy` fetches a proxy object.
- The `as_string()` method converts the proxy into a format compatible with `httpx`.
- For synchronous requests, use `httpx.Client`.
- For asynchronous requests, use `httpx.AsyncClient`.
- The `proxies` parameter is used to route requests through the proxy.

Note

If the proxy is working correctly, the output should be an `IPv4` address that is not your own.

For more advanced use cases, such as caching and automatic rotation, consider using the `ProxyInterface` class. For additional details, visit the References page.

Requests Library

`swiftshadow` integrates seamlessly with the popular `requests` library, allowing you to easily route your HTTP/HTTPS requests through a proxy.

Example Usage

Here’s how you can use a proxy fetched by `QuickProxy` with the `requests` library:

```
from swiftshadow import QuickProxy
from requests import get

# Fetch a proxy
proxy = QuickProxy()

# Use the proxy with requests
resp = get('https://checkip.amazonaws.com',
proxies=proxy.as_requests_dict())
print(resp.text)
```

Explanation

- `QuickProxy` fetches a proxy object.

- The `as_requests_dict()` method converts the proxy into a format compatible with the `requests` library.
- The `proxies` parameter in `requests.get()` is used to route the request through the proxy.

Note

If the proxy is working correctly, the output should be an `IPv4` address that is not your own.

For more advanced use cases, such as caching and automatic rotation, consider using the `ProxyInterface` class. For additional details, visit the References page.