Python-Amazon-MWS

Release 1.0dev16

Galen Rice, Alex Hellier, James Hiew

Sep 13, 2023

CONTENTS

python-amazon-mws is a Python connector to Amazon Marketplace Web Services (or MWS). It provides a simple way to build and send requests to MWS, allowing access to all that MWS can do from your Python application.

Use Feeds to update your product listings, run Reports, get updates on your Orders, create and manage FBA Inbound Shipments... do it all with python-amazon-mws!

Note: This is a third-party connector with no direct affiliation to Amazon.

CHAPTER

ONE

CHANGELOG

1.1 v1.0dev16

Note: This is a **prerelease** version for **v1.0**.

Date November 2020

Issues See here

This update focused on the InboundShipments API, adding some new ways to input and manage data related to FBA shipments while also introducing some comprehensive documentation of the same.

Also includes the Products API's *get_my_fees_estimate* method, deprecation warnings for old argument names to smooth the transition from v0.8.

1.1.1 Major changes

- Products API *get_my_fees_estimate* method added.
  - See #216 for details.
- Deprecation warnings for old argument names.
  - Some argument names for certain requests had changed between v0.8 and v1.0dev. This change makes it possible to use the v0.8 argument names in current code.
  - When using an old argument name, the method will raise a deprecation warning, indicating those old argument names will be removed in v1.1. The method will then proceed as expected using the correct arg names.
  - See #222 for details.
- Datatype models added for InboundShipments.
  - All models for this API can be found in module `mws.models.inbound_shipments`.
  - Added datatype models for `Address`, `PrepDetails <mws.InboundShipments.PrepDetails>`, `InboundShipmentPlanRequestItem <mws.InboundShipments.InboundShipmentPlanRequestItem>`, and `InboundShipmentItem <mws.InboundShipments.InboundShipmentItem>`. These models can be used in relevant arguments for request methods related to FBA shipment creation and updating (*create_inbound_shipment_plan*, *create_inbound_shipment*, and *update_inbound_shipment*.

    * With this addition, it is now possible to include `PrepDetails` for items being added to shipments. This was not possible using the now-"legacy" item dictionary method (though it is still possible using the lower-level generic requests).

  – Added `shipment_items_from_plan` helper method.

    * The method can process the contents of a shipment plan from the parsed response from `create_inbound_shipment_plan`, turning the returned items into a set of `InboundShipmentItem` models automatically.

- **New documentation for Managing FBA Shipments added.**
  – See: *Managing Fulfillment Inbound (FBA) Shipments*.
  – Comprehensive documentation for how to manage FBA shipments using the InboundShipments API.
  – Showcases the usage of new models provided by this update.

1.1.2 Minor changes

- Links to Amazon MWS documentation throughout the code base updated from *http://* to *https://*.
- Type annotations added to request methods for InboundShipments API. - As part of this, certain *assert*-style checks for argument types have been removed.
- Tests for InboundShipments request methods overhauled, removing dependency on *unittest* in favor of *pytest*.
- URL naming improvements for documentation pages, and proper usage of doc links instead of adding extraneous anchor links. - Some bookmarks may break with this change, apologies!
- Dev update callout removed from project README.
- Development tooling configurations moved into *setup.cfg* for consistency.
- Project test suite expanded to Python 3.9 and Ubuntu-20.04 - All automated testing is already performed in a matrix strategy, across Python 3.6, 3.7, 3.8, and 3.9; and on OSes Windows, MacOS, Ubuntu-18, and Ubuntu-20. Every combination of all these versions and OSes is tested.

1.2 v1.0dev15

Note: This is a **prerelease** version for **v1.0**.

Date September 2020

This update represents a major step towards a v1.0 release candidate. Much of the core components of the project have been restructured, new XML parsing logic has been added, and API code has been streamlined to ease development efforts going forward.

1.2.1 Major changes

- Added dependency `xmltodict` for parsing XML documents to Python dict objects.
- Added `MWSResponse`, intended to replace `DictWrapper` and `DataWrapper` response wrappers; and `DotDict`, intended to replace `ObjectDict`.
  - These features are in preview mode. See[1] .
- Added `MWS.generic_request()`, a low-level interface for sending requests to any MWS operation with any set of parameters necessary (using new utility function, `mws.utils.params.flat_dict_param`).
- Several objects have been moved, renamed, and/or retooled to improve code structuring and interoperability, most notably `mws.utils` (which has been broken down to multiple modules with different concerns).

1.2.2 Code restructuring

Several objects have been moved and/or renamed, with new modules added to contain them. At the same time, the namespace for most of those objects has been left mostly intact. Following changes relate to objects whose imports may need to be adjusted in application code.

- `mws.utils`, formerly a single module file, is now a directory containing other modules with separated concerns.
- Moved `mws.mws.DictWrapper` to `mws.utils.DictWrapper`.
- Moved `mws.mws.DataWrapper` to `mws.utils.DataWrapper`.
- Moved `mws.mws.ObjectDict` to `mws.utils.ObjectDict`.
- Moved `mws.mws.XML2Dict` to `mws.utils.XML2Dict`.
- Moved/renamed `mws.mws.clean_params` to `mws.utils.clean_params_dict`.
  - "Cleaning" logic has been broken down further with `mws.utils.params.clean_value`, which passes to other "clean_FOO" methods such as `clean_string`, `clean_bool`, and `clean_date`.
- Changed `mws.utils.get_utc_timestamp` to `mws.utils.mws_utc_now`.
  - `get_utc_timestamp` returned an ISO-8601-formatted string of the current datetime in UTC timezone. `mws_utc_now` produces the same datetime, but instead returns a `datetime.datetime` object. An ISO-8601 formatted string can easily be obtained using the `.isoformat()` method.
- Changed `mws.mws.remove_namespace` to `mws.utils.remove_xml_namespaces`.
  - The new version works the same as the old, but can now accept bytes as well as strings.

[1] **1.0dev15 features preview**: Prior to **v1.0**, `DictWrapper` and `DataWrapper` will still be used as default response wrappers for all requests; and the `.parsed` interface for these objects will continue to be `ObjectDict` instances.

To use `MWSResponse` and `DotDict` for response parsing in development versions (1.0dev15 and up), you must enable the `_use_feature_mwsresponse` feature flag:

1. Instantiate an API class, i.e. `feeds_api = Feeds(...)`.
2. Set flag `_use_feature_mwsresponse` to `True` on the class instance: `feeds_api._use_feature_mwsresponse = True`.

Now all requests made through this class instance will return responses as `MWSResponse`.

1.2.3 Deprecations

The following have been **deprecated**:

- `DictWrapper` (removed in v1.1), replaced by `MWSResponse` in v1.0 (currently in preview mode).
- `DataWrapper` (removed in v1.1), replaced by `MWSResponse` in v1.0 (currently in preview mode).
- `ObjectDict` (removed in v1.1), replaced by `DotDict` in v1.0 (currently in preview mode).
- `XML2Dict` (removed in v1.1). XML parsing into Python objects will be performed by the `xmltodict` library starting in v1.0.
- `MWS.enumerate_param` (removed in v1.0). Use utility methods found in `mws.utils.params`, instead.

1.2.4 Minor changes

- New arguments are available when instantiating an API class (subclasses of the `MWS` main class, such as `Feeds` and `Orders`):
  - Argument and class attr `user_agent_str` sets the User Agent String sent with requests to MWS. This can be used to override PAM's default agent string, `"python-amazon-mws/{version} (Language=Python)"`.
  - Argument `headers` and attribute `extra_headers` accepts a dictionary with headers to add to each request, if necessary. Headers can still be altered per-request by passing an `extra_headers` kwarg to `make_request` or `generic_request`.
  - Argument and class attr `force_response_encoding` allows specifying the encoding used to decode a response's bytes content, when parsed by `MWSResponse` into a `DotDict`.
    - Amazon documentation states they use ISO-8859-1 (aka Latin-1) encoding. However, some responses may still be encoded differently, such as in UTF-8, even if this behaviour is not well-documented. By default, python-amazon-mws relies on `requests.Response.apparent_encoding` to guess the character set to decode, which should be sufficient for most uses.
    - Setting `force_response_encoding='utf-8'`, for example, will force responses to be decoded as UTF-8 automatically for any request made with that API class instance.
    - Encoding can also be adjusted on the `MWSResponse` object, by assigning `response.encoding = 'utf-8'` and then calling `response.parse_response()` to re-parse content.
- All request methods are now required to pass the `Action` name of an MWS operation as the first argument to `MWS.make_request` or `MWS.generic_request`. Previously, this was expected as a parameter in the data sent with a request.
- `MWS.make_request` argument `extra_data` has been renamed to `params`, and can now default to `None`. This permits operations such as `GetServiceStatus`, which require no parameters, to pass without issue.
- The `timeout` kwarg in `MWS.make_request` has been promoted to a named argument, with a default value of 300 seconds.

CHAPTER

TWO

PREREQUISITES FOR MWS CONNECTIVITY

See also:

All links in this documentation point to the `developer.amazonservices.com` domain, but other regional domains are provided by Amazon. For a list of portals for other regions, please see Related Resources (MWS documentation link).

In order to use python-amazon-mws, you must have an Amazon Professional Seller account, and you must register as a developer. You will then be provided a set of MWS credentials, which include your **Seller ID**, **Access Key**, and **Secret Key** (and, possibly, **Auth Token**).

These credentials, along with a Marketplace ID, will be needed to make requests to MWS, whether using python-amazon-mws or any other MWS-related service.

2.1 Test MWS access using Scratchpad

You can test your access to MWS using **Amazon MWS Scratchpad** (docs):

1. Open the Scratchpad.

   > **Warning:** Always verify the URL of the Scratchpad before entering your MWS credentials! The domain should be `mws.amazonservices.com` or one of Amazon's other regional domains (see here for a list of regional portals).

2. Enter your **MWS credentials** in the **Authentication** section.
3. In **API Section**, choose "Products".
4. In **Operation**, choose "ListMatchingProducts".
5. Under **Required API Parameters**, enter:
   - `MarketplaceID`: A valid MarketplaceID for your desired marketplace (example: `ATVPDKIKX0DER` for the US market). See: Amazon MWS endpoints and MarketplaceId values.
   - `Query`: `python`, to search for products containing "python" somewhere in their description.
6. Click **Submit**.

If your access works, you should an XML response beginning with `ListMatchingProductsResponse`. Otherwise, you may see an `ErrorResponse`, with an error message indicating the problem.

CHAPTER

THREE

INSTALLATION

Currently, two versions of the package are available: an older **v0.8.x**, available on PyPI; and the in-development **v1.0devXY**, available on GitHub.

For new projects, we recommend using **v1.0devXY**, as it contains a more complete set of API sections and operations. Note that this version is still pre-alpha, so some parts of the package are subject to change as we slowly update from the original 0.8.x code.

Use `pip` to install 1.0devXY from GitHub off the `develop` branch:

```
pip install git+https://github.com/python-amazon-mws/python-amazon-mws.git@develop
↪#egg=mws
```

CHAPTER

FOUR

GETTING STARTED

Note: We assume you have an Amazon Professional Seller account and developer access to be able to use MWS. If not, please see *Prerequisites for MWS connectivity*.

4.1 Entering credentials

To begin, use your MWS Credentials to instantiate one of the API classes. We will use the `Products` API for this example.

Where you store these credentials is up to you, but we recommend using environment variables, like so:

```
import os
from mws import Products

products_api = Products(
    access_key=os.environ["MWS_ACCESS_KEY"],
    secret_key=os.environ["MWS_SECRET_KEY"],
    account_id=os.environ["MWS_ACCOUNT_ID"],
    auth_token=os.environ["MWS_AUTH_TOKEN"],
)
# `auth_token` is optional, depending on how you your MWS access is set up.
```

4.2 Making requests

Each API class contains a number of **request methods**, which closely match the Operations available to that API section in MWS. You should refer to MWS documentation for the API class you intend to use and provide the data specified by that operation.

For our example, we will use the Products API operation ListMatchingProducts. In python-amazon-mws, this is done using an instance of the `Products` API class and its method `list_matching_products`:

```
from mws import Marketplaces

# Marketplaces is an enum we can use to fill in the `marketplace_id` value,
# instead of needing to manually enter, i.e., "ATVPDKIKX0DER"
my_marketplace = Marketplaces.US.marketplace_id

response = products_api.list_matching_products(
```

(continues on next page)

(continued from previous page)

```
    marketplace_id=my_marketplace,
    query="python",
)
```

The request is sent automatically when `list_matching_products` is called, and a `response` is returned. MWS typically returns an XML document encoded in ISO-8859-1 (per Amazon's standards), which python-amazon-mws attempts to decode automatically.

For most responses (including our example `list_matching_products`), the `response` will be a `DictWrapper` object containing:

- `response.original`, the original XML document;
- `response.response`, the HTTP response code of the request (200, 400, etc.); and
- `response.parsed`, a parsed version of the XML tree. (See *Using Parsed XML Responses*).

Certain responses (such as the GetReport operation, under the Reports API) may return other content types, such as PDFs, tab-delimited flat files, ZIP files, and so on. Non-XML responses will be wrapped in a `DataWrapper` object with similar attributes as `DictWrapper`, with the raw document stored in `.original`, and `.parsed` simply returning `.original` for convenience.

Warning: New in version 1.0dev15.

`DictWrapper` and `DataWrapper` are deprecated, and will be removed in v1.1. During development testing, these objects will still be returned from requests by default, and parsed content will still use `ObjectDict` instances (also deprecated).

To use newer features, such as the *`MWSResponse`* wrapper and parsed XML using *`DotDict`*, set flag `_use_feature_mwsresponse` to `True` on an API class instance *before* making any requests:

```
# instantiate your class
products_api = Products(...)

# set the new feature flag
products_api._use_feature_mwsresponse = True

# run your requests as normal
response = products_api.list_matching_products(...)
```

For details on using these newer features, please see:

- *Using Parsed XML Responses*
- *MWSResponse*
- *DotDict*

`MWSResponse` *and* `DotDict` *will become the default objects returned by requests in v1.0.*

CHAPTER

FIVE

GENERIC REQUESTS

New in version 1.0dev15: Generic request support added.

While most MWS operations are well-covered by python-amazon-mws with dedicated and purpose-built request methods, Amazon may occasionally update MWS to include new parameters that we do not yet provide access to. Either that, or you just want lower-level access to input your own request, without going through the rest of python-amazon-mws to do so.

For these situations, you can use `APIClass.generic_request()`, available in all API classes that inherit from the base `MWS` class.

5.1 Back to basics

To use `.generic_request()`, you must first instantiate the API class that contains the operation you want to send. Using the correct API class is required, as the base URI used the build the request is different for each API section. For instance, to use the `ListOrders` operation in the `Orders` API, you would create an `Orders` instance.

With the class instantiated, specify the operation to call as the `action` arg to `.generic_request()`; then provide a dict of parameters for your request as `params`:

```
import datetime

from mws import Orders, Marketplaces

my_marketplace_ids = [
    Marketplaces.US.marketplace_id,
    Marketplaces.UK.marketplace_id,
]

orders_api = Orders(MY_ACCESS_KEY, MY_SECRET_KEY, MY_ACCOUNT_ID)

response = orders_api.generic_request(
    action="ListOrders",
    params={
        "MarketplaceId.Id": my_marketplace_ids,
        "CreatedAfter": datetime.datetime(2020, 8, 28),
    }
)
```

The above is equivalent to calling `Orders.list_orders` with:

```
response = orders_api.list_orders(
    marketplace_ids=my_marketplace_ids,
```

(continues on next page)

(continued from previous page)

```
    created_after=datetime.datetime(2020, 8, 28),
)
```

Key differences between a generic request and the "pythonic" version include:

- The `action` must be specified for each call, using the case-sensitive name of the MWS operation (usually in CapCase with no underscores).
- `params` must include case-sensitive keys matching the parameters required for the MWS operation, according to Amazon documentation.
- The `params` dict is *flattened*, such that nested lists and dicts in `params` are keyed and enumerated into appropriate request parameter keys.

5.2 Parameter dict flattening

Generic requests make use of *`flat_param_dict()`* to convert nested Mappings and Iterables into a "flat" set of key-value pairs.

Rules

- Nested mapping objects (`dict`, `DotDict`, etc.) are recursively flattened, joining the keys of the child mapping to the parent key with `.`.
- Nested iterables (`list`, `tuple`, `set`, etc.) are enumerated with a 1-based index, with each index joined to the parent key with `.`.
- All nested mappings and iterables are processed recursively, flattening other mappings and iterables along the way.

Example

```
value = {
    "a": 1,
    "b": "hello",
    "c": [
        "foo",
        "bar",
        {
            "spam": "ham",
            "eggs": [
                5,
                6,
                7,
            ],
        },
    ],
}
```

The above, when passed through `flat_param_dict()`, produces:

```
{
    "a": 1,
    "b": "hello",
    "c.1": "foo",
    "c.2": "bar",
    "c.3.spam": "ham",
    "c.3.eggs.1": 5,
    "c.3.eggs.2": 6,
    "c.3.eggs.3": 7,
}
```

- "a" and "b" keys point to non-dict, non-sequence values (not including strings), so they return their original values.
- "c" contains an iterable (list), which is enumerated with a 1-based index. Each index is concatenated to "c" with ".", creating keys "c.1" and "c.2".
- At "c.3", another nested object was found. This is processed recursively, and each key of the resulting dict is concatenated to the parent "c.3" to create multiple keys in the final output.
- The same occurs for "c.3.eggs", where an iterable is found and is enumerated.
- The final output should always be a flat dictionary with key-value pairs.

Using a prefix

`flat_param_dict` accepts a `prefix` argument, used mainly when flattening nested objects recursively. When provided, all keys in the resulting output will begin with `prefix + '.'`:

```
# Using the same `value` as before:
flat_param_dict(value, prefix="example")

# Produces:
{
    "example.a": 1,
    "example.b": "hello",
    "example.c.1": "foo",
    "example.c.2": "bar",
    "example.c.3.spam": "ham",
    "example.c.3.eggs.1": 5,
    "example.c.3.eggs.2": 6,
    "example.c.3.eggs.3": 7,
}
```

5.3 Generic request component methods

`MWS.`**`generic_request`**(*action*, *params=None*, *method='POST'*, *timeout=300*, ***kwargs*)

Builds a generic request with arbitrary parameter arguments. This method should be called from an API subclass (`Orders`, `Feeds`, etc.), else the `uri` attribute of the class instance must be set manually.

This method's signature matches that of `.make_request`, as the two methods are similar. However, `params` is expected to be either the default `None` or a nested dictionary, that is then passed to *`flat_param_dict()`*.

`mws.utils.params.`**`flat_param_dict`**(*value*, *prefix=''*)

Returns a flattened params dictionary by collapsing nested dicts and non-string iterables.

Any arbitrarily-nested dict or iterable will be expanded and flattened.

- Each key in a child dict will be concatenated to its parent key.
- Elements of a non-string iterable will be enumerated using a 1-based index, with the index number concatenated to the parent key.
- In both cases, keys and sub-keys are joined by `.`.

If `prefix` is set, all keys in the resulting output will begin with `prefix + '.'`.

Parameters

- **value** (*Union[str, collections.abc.Mapping, List]*) –
- **prefix** (*str*) –

Return type dict

CHAPTER

SIX

MANAGING FULFILLMENT INBOUND (FBA) SHIPMENTS

Warning: The following includes features added in **v1.0dev16** related to Datatype models. Models can be called from the API class that uses them. For example, to use the `Address` model attached to the *`InboundShipments`* API:

```
from mws import InboundShipments

# from the class itself:
my_address = InboundShipments.Address(...)

# or from an instance of the class:
inbound_shipments_api = InboundShipments(...)
my_address = inbound_shipments_api.Address(...)
```

Note: Examples in this document use *MWSResponse preview features*.

MWS handles **Fulfillment Inbound Shipments**, also known as **FBA** (for "Fulfillment By Amazon") through the Fulfillment Inbound Shipment API section. Users should familiarize themselves with this section of the API in MWS documentation before getting started.

In python-amazon-mws, this API is covered by *`InboundShipments`*.

6.1 Basic steps to create a shipment in MWS

For a quick overview, MWS requires the following pattern to creating FBA shipments:

1. Send a request to *`create_inbound_shipment_plan`* with all items you wish to ship, along with their quantities, conditions, prep details, and so on.
2. MWS will respond with one or more **shipment plans**, indicating where to send each of your items. Multiple shipments may be requested, and the same item may have its quantities split between these shipments. Each plan also returns the FBA Shipment ID needed to create a shipment, as well as the ID and address of the Fulfillment Center that will expect that shipment.
3. For each shipment plan, send a *`create_inbound_shipment`* request with the items, quantities, and other details identified in the plan.
    - Optionally, it is possible to use *`update_inbound_shipment`* to add planned items for a new shipment to an existing shipment under certain conditions. **Using this option improperly may violate the terms of your seller account, so use with caution!**

We'll look at each of these steps in detail below.

> **Warning:** MWS does not provide a sandbox for testing functionality. If you use examples from this guide for testing purposes, you will need to use **live data** to do it, and will be creating **real FBA shipments**. Please use this guide at your own risk.
>
> Some things to keep in mind when testing this functionality:
>
> - Make note of any Shipment IDs for shipments you generate with these examples.
> - Use custom shipment names to help identify test shipments, such as "TEST_IGNORE", so you can more easily find those shipments in Seller Central, if you lose track of them in testing.
> - Inform other members of your organization that you are conducting tests, particularly if they use Seller Central or other MWS-related tooling to check on shipment statuses.
> - Leaving test shipments in WORKING or SHIPPED statuses may have an impact on your product inventory. We advise changing these to CANCELLED when you complete your testing.

6.2 Requesting a shipment plan

We start by informing Amazon we have items we wish to ship, requesting a **shipment plan** through MWS.

You will need:

- MWS credentials to authenticate with MWS (not in scope for these docs).
- A valid **ship-from address**, presumably the address of the facility where you will be shipping items from.
- A list of Seller SKUs for items in your product catalog to add to new shipments.

6.2.1 Create the API instance

To begin, create an instance of `InboundShipments` as you would any other API class in python-amazon-mws. You will then use this API class instance to initiate requests to MWS.

```
from mws import InboundShipments

# assuming MWS credentials are stored in environment variables (your setup may vary):
inbound_api = InboundShipments(
    access_key=os.environ("MWS_ACCESS_KEY"),
    secret_key=os.environ("MWS_SECRET_KEY"),
    account_id=os.environ("MWS_ACCOUNT_ID"),
)
```

6.2.2 Create your ship-from address

Next, set up your ship-from address, which is required for the three core operations related to FBA shipments: planning, creation, and updating.

The simplest way to store your ship-from address is to create an instance of the `Address` model:

```
my_address = inbound_api.Address(
    name="My Warehouse",
    address_line_1="555 Selling Stuff Lane",
    address_line_2="Suite 404",
    city="New York",
    district_or_county="Brooklyn",
    state_or_province_code="NY",
    country_code="US",
    postal_code="11265",
)
```

This model closely follows the structure of MWS's Datatype of the same name. You should refer to MWS documentation for this Datatype to ensure all necessary elements of your address are included.

Note: If you're curious, you can use any model's `.to_params()` method to return a dictionary containing the request parameters of that model and their values.

```
my_address.to_params()
# {'Name': 'My Warehouse', 'AddressLine1': '555 Selling Stuff Lane', 'AddressLine2':
↪'Suite 404', 'City': 'New York', 'DistrictOrCounty': 'Brooklyn',
↪'StateOrProvinceCode': 'NY', 'CountryCode': 'US', 'PostalCode': '11265'}
```

This method also accepts a `prefix` argument, which adds the prefix string plus `'.'` before each parameter key:

```
my_address.to_params("ShipFromAddress")
# {'ShipFromAddress.Name': 'My Warehouse', 'ShipFromAddress.AddressLine1': '555␣
↪Selling Stuff Lane', 'ShipFromAddress.AddressLine2': 'Suite 404', 'ShipFromAddress.
↪City': 'New York', 'ShipFromAddress.DistrictOrCounty': 'Brooklyn', 'ShipFromAddress.
↪StateOrProvinceCode': 'NY', 'ShipFromAddress.CountryCode': 'US', 'ShipFromAddress.
↪PostalCode': '11265'}
```

Using `.to_params()` in your own code is usually not necessary, as most request methods will convert the model instance to parameters automatically.

Optional: Store your ship-from address on the API instance

If you plan to make several requests in a row related to the same ship-from address, you can store the address on an instance of `InboundShipments` API as `.from_address`:

```
inbound_api.from_address = my_address
```

When using this option, you can omit passing `from_address=my_address` as an argument in the request examples below. All relevant request methods (`create_inbound_shipment_plan`, `create_inbound_shipment`, and `update_inbound_shipment`) will pass the stored `from_address` to these requests automatically.

In any case, supplying a `from_address` argument to one of these methods will be used as an override, regardless of the address stored within the API instance.

6.2.3 Request a shipment plan

Amazon's workflow for creating a shipment uses the following pattern:

1. Create a **shipment plan** by sending a `CreateInboundShipmentPlan` request. This informs Amazon which items you intend to ship and the total quantity for each, as well as any prep details, item conditions, and so on.
2. MWS responds with one or more planned shipments for those items. They may request certain items are sent to certain fulfillment centers, and may even split quantities for some items to multiple facilities. You must use the planned shipments to create your actual shipments.
3. Send a `CreateInboundShipment` request for *each* planned shipment. This should include the ShipmentId, DestinationFulfillmentCenterId, and any items and quantities returned in the response from `CreateInboundShipmentPlan`, so that the new shipment matches the planned one.
4. A successful request to `CreateInboundShipment` will create an FBA Shipment, which you can further interact with through MWS or on Seller Central.

We'll start by creating the shipment plan, for which we need a list of items.

Building a list of planned items

Each item in your shipment plan can be represented by an instance of `InboundShipmentPlanRequestItem`, which closely follows the MWS Datatype of the same name:

```
item1 = inbound_api.InboundShipmentPlanRequestItem('MY-SKU-1', 36)
item2 = inbound_api.InboundShipmentPlanRequestItem('MY-SKU-2', 12)

my_items = [item1, item2]
```

The only required arguments for the model are `sku` and `quantity`, which are sufficient for loose item shipments of new items when prep details do not need to be specified.

Note: You can add more detail to an `InboundShipmentPlanRequestItem` instance, depending on your needs. If you were sending, for example, an item that comes in case-packs of 12, in NewOEM condition, with a particular ASIN, and requires Amazon to prep each item with Polybagging; you might create that item model like so:

```
my_condition = inbound_api.ItemCondition.NEW_OEM  # or the string "NewOEM"
my_prep_details = inbound_api.PrepDetails(
    prep_instruction=PrepInstruction.POLYBAGGING,  # or "Polybagging"
    prep_owner=PrepDetails.AMAZON  # or "AMAZON"
)

detailed_item = inbound_api.InboundShipmentPlanRequestItem(
    sku='MY-OTHER-SKU',
    quantity=48,
    quantity_in_case=12,
    asin='B0123456789',
    condition=my_condition,
    prep_details_list=[my_prep_details],
)
```

Again for the curious, `detailed_item.to_params()` looks like so:

```
detailed_item.to_params()
# {'SellerSKU': 'MY-OTHER-SKU', 'ASIN': 'B0123456789', 'Condition': 'NewOEM',
↪'Quantity': 48, 'QuantityInCase': 12, 'PrepDetailsList.member.1.PrepInstruction':
↪'Polybagging', 'PrepDetailsList.member.1.PrepOwner': 'AMAZON'}
```

(continues on next page)

(continued from previous page)

Sending the request

Now that we have our items handy, it's time to make our request for a shipment plan:

```
# using `inbound_api`, `my_address` and `my_items` from previous examples
resp = inbound_api.create_inbound_shipment_plan(my_items, from_address=my_address)
```

Other arguments you can provide include:

- `country_code` *or* `subdivision_code`, the country or country subdivision you are planning to send a shipment to. `country_code` defaults to `"US"`; `subdivision_code` (which refers to a subdivision of India specifically) defaults to `None`.
  - According to MWS documentation, providing both options will return an error.
- `label_preference`, a preference for label preparation. Defaults to `None`, which MWS may interpret as "SELLER_LABEL" internally.

And note that the `from_address` argument is optional if the address has been *stored on the API instance*.

6.3 Processing shipment plans

If our request to create shipment plans was successful, MWS will respond with an XML document containing plan details. python-amazon-mws will *automatically parse this response*, giving us access to the Python representation of the response in `resp.parsed`.

For reference, we will use the following example XML response from `create_inbound_shipment_plan`. You can access this document in your own response by checking `resp.original.text`:

```
<?xml version="1.0"?>
<CreateInboundShipmentPlanResponse
  xmlns="http://mws.amazonaws.com/FulfillmentInboundShipment/2010-10-01/">
  <CreateInboundShipmentPlanResult>
    <InboundShipmentPlans>
      <member>
        <DestinationFulfillmentCenterId>ABE2</DestinationFulfillmentCenterId>
        <LabelPrepType>SELLER_LABEL</LabelPrepType>
        <ShipToAddress>
          <City>Breinigsville</City>
          <CountryCode>US</CountryCode>
          <PostalCode>18031</PostalCode>
          <Name>Amazon.com</Name>
          <AddressLine1>705 Boulder Drive</AddressLine1>
          <StateOrProvinceCode>PA</StateOrProvinceCode>
        </ShipToAddress>
        <EstimatedBoxContentsFee>
          <TotalUnits>10</TotalUnits>
          <FeePerUnit>
            <CurrencyCode>USD</CurrencyCode>
            <Value>0.10</Value>
          </FeePerUnit>
```

(continues on next page)

(continued from previous page)

```
          <TotalFee>
            <CurrencyCode>USD</CurrencyCode>
            <Value>10.0</Value>
          </TotalFee>
        </EstimatedBoxContentsFee>
        <Items>
          <member>
            <FulfillmentNetworkSKU>FNSKU00001</FulfillmentNetworkSKU>
            <Quantity>1</Quantity>
            <SellerSKU>SKU00001</SellerSKU>
            <PrepDetailsList>
              <PrepDetails>
                <PrepInstruction>Taping</PrepInstruction>
                <PrepOwner>AMAZON</PrepOwner>
              </PrepDetails>
            </PrepDetailsList>
          </member>
          <member>
            ...
          </member>
        </Items>
        <ShipmentId>FBA0000001</ShipmentId>
      </member>
      <member>
        ...
      </member>
    </InboundShipmentPlans>
  </CreateInboundShipmentPlanResult>
  <ResponseMetadata>
    <RequestId>babd156d-8b2f-40b1-a770-d117f9ccafef</RequestId>
  </ResponseMetadata>
</CreateInboundShipmentPlanResponse>
```

6.3.1 Gathering shipment details

To begin, we can access each shipment plan in the parsed response like so:

```
# Using the `resp` object from our previous examples
for plan in resp.parsed.InboundShipmentPlans.member:
    ...
```

Each `plan` contains metadata required for creating a new shipment. These include:

- `plan.ShipmentId`, the FBA shipment ID Amazon generates for the new shipment plan.
- `plan.DestinationFulfillmentCenterId`, the short code for a Fulfillment Center planning to receive this shipment.
- `plan.LabelPrepType`, the label preparation type for this shipment.

In addition to these data points, you should consider gathering the following data as arguments for the `create_inbound_shipment` request method:

- `shipment_name` (required), a human-readable name to help identify your shipment without relying on shipment IDs.
- `shipment_status`, the initial status of the shipment. Defaults to "WORKING", indicating the shipment will remain "open" so that items and quantities can still be changed before it is shipped.

The following constants can be used for this argument:

- `InboundShipments.STATUS_WORKING`
- `InboundShipments.STATUS_SHIPPED`
- `InboundShipments.STATUS_CANCELLED`
- `InboundShipments.STATUS_CANCELED` (alias for `STATUS_CANCELLED`)

- `case_required`, a boolean indicating that items in the shipment are either *all case-packed* (if `True`) or *all loose items* (if `False`). Defaults to `False`.
- `box_contents_source`, a string indicating a source of box content data for packages within the shipment, or `None` indicating no box contents source. Defaults to `None`.

  The following constants can be used for this argument:

  - `InboundShipments.BOX_CONTENTS_FEED`, indicating contents will be provided in a *Feed* of type `_POST_FBA_INBOUND_CARTON_CONTENTS_`.
  - `InboundShipments.BOX_CONTENTS_2D_BARCODE`, indicating contents will be provided using 2D barcodes present on the cartons of the shipment.

We will illustrate how to use these data points later in this doc.

6.3.2 Converting plan items to shipment items

While the request to `create_inbound_shipment_plan` makes use of the `InboundShipmentPlanRequestItem` model to transmit item data, this model is not sufficient for passing data to `create_inbound_shipment` and `update_inbound_shipment` requests, as they require slightly different parameters. We will need to use the `InboundShipmentItem` model, instead.

We can pass data to this model in one of three ways:

1. Manually processing item data from the response:

```
for plan in resp.parsed.InboundShipmentPlans.member:
    shipment_items = []
    for item in plan.Items.member:
        new_item = inbound_api.InboundShipmentItem(
            sku=item.SellerSKU,
            quantity=item.Quantity,
        )
        shipment_items.append(new_item)
```

2. Using `InboundShipmentItem.from_plan_item` to construct an item automatically from each item in the response:

```
for plan in resp.parsed.InboundShipmentPlans.member:
    shipment_items = []
    for item in plan.Items.member:
        new_item = inbound_api.InboundShipmentItem.from_plan_item(item)
        shipment_items.append(new_item)
```

3. Using helper method `shipment_items_from_plan` to return a list of items from the entire plan automatically:

```
for plan in resp.parsed.InboundShipmentPlans.member:
    shipment_items = inbound_api.shipment_items_from_plan(plan)
```

Note: Using `InboundShipmentItem.from_plan_item` or `shipment_items_from_plan`, each item will automatically store the `fnsku` of each planned item. This data is ignored in calls to `create_inbound_shipment` and `update_inbound_shipment`, but can be useful for tracking items internally.

Using either of these methods, the list of `shipment_items` can be used as the `items` argument to either the `create_inbound_shipment` or `update_inbound_shipment` request method.

Adding `quantity_in_case` and `release_date` values

Item data provided by a `plan` is sufficient for most data required for items, but some data points must be added manually:

- Case-pack information, specifically the `quantity_in_case` argument, is not supplied by the response from `create_inbound_shipment_plan`, even if this information was provided in the request itself.
- Pre-order items must provide an additional `release_date` data point.

In the first two examples *above*, these data points can be added as arguments when constructing the new item:

```
# using InboundShipmentItem(...):
new_item = inbound_api.InboundShipmentItem(
    sku=item.SellerSKU,
    quantity=item.Quantity,
    quantity_in_case=...,
    release_date=...,
)

# using InboundShipmentItem.from_plan_item(...):
new_item = inbound_api.InboundShipmentItem.from_plan_item(
  item,
  quantity_in_case=...,
  release_date=...,
)

# Confirm this data has been added:
print(new_item.quantity_in_case, new_item.release_date)
```

In either case, when working with multiple items per shipment plan, you will need to determine which SKU these data refer to. You should be able to rely on `item.SellerSKU` to identify those SKUs.

Adding extra data when processing items in bulk

When processing a planned shipment's items in bulk, adding `quantity_in_case` and/or `release_date` values to each item can be done using the `overrides` argument to `shipment_items_from_plan`.

`overrides` expects a dictionary with SellerSKUs as its keys. The values of this dict can be either:

- A dict containing keys `quantity_in_case` and/or `release_date` (all other keys are ignored):

```
overrides = {
    'mySku1': {
        'quantity_in_case': 12,
        'release_date': datetime.datetime(2020-12-25),
```

(continues on next page)

(continued from previous page)

```
    },
}
```

- An instance of `ExtraItemData`:

```
overrides = {
    'mySku2': inbound_api.ExtraItemData(
        quantity_in_case=12,
        release_date=datetime.datetime(2020-12-25),
    ),
}
```

You should construct this set of overrides for all SKUs sent in your original request to `create_inbound_shipment_plan`. You can then use the same set of overrides on any planned shipment resulting from that request:

```
overrides = {...}

for plan in resp.parsed.InboundShipmentPlans.member:
    shipment_items = inbound_api.shipment_items_from_plan(plan, overrides=overrides)
```

6.4 Creating shipments

Putting everything together up to this point, we can create a new FBA shipment using the *`create_inbound_shipment`* method:

```
# with optional overrides
overrides = {
    'mySku1': inbound_api.ExtraItemData(...),
    'mySku2': inbound_api.ExtraItemData(...),
}

for plan in resp.parsed.InboundShipmentPlans.member:
    # Gather our items for the planned shipment
    shipment_items = inbound_api.shipment_items_from_plan(plan, overrides=overrides)

    # Send the request to create a new shipment
    new_shipment_resp = inbound_api.create_inbound_shipment(
        shipment_id=plan.ShipmentId,
        shipment_name="My Shiny New FBA Shipment",
        destination=plan.DestinationFulfillmentCenterId,
        items=shipment_items,
        label_preference=plan.LabelPrepType,
    )
```

For help with additional arguments - such as `shipment_status`, `case_required`, `box_contents_source`, or `from_address` - see *Gathering shipment details*.

6.5 Updating shipments

Creating a shipment is not the end of the story, of course. It is sometimes necessary to make changes to an already-created shipment. For this, we use *update_inbound_shipment*.

`update_inbound_shipment`'s arguments are identical to those of `create_inbound_shipment`, with the exception that all arguments besides `shipment_id` are optional. Generally, supplying a value to one of those arguments will overwrite that value of the given shipment, such as:

- Setting `shipment_status=InboundShipments.STATUS_CANCELLED` to cancel a shipment;
- Changing the `from_address`;
- etc.

6.5.1 Changing item quantities

Item quantities on a shipment can be changed by providing a list of `InboundShipmentItem` instances for the `items` argument of `update_inbound_shipment`. The details of the submitted items will overwrite details of those items in the existing shipment based on matching SellerSKUs.

Amazon will expect the *total* quantity for an item: there is no mechanism for adding or subtracting a quantity from the existing total. For example, if a shipment contains **24** units of an item and you want to add **12** of that item, you will need to submit a total quantity of **36** in the update request:

```
resp = inbound_api.update_inbound_shipment(
    shipment_id="FBAMYSHIPMENT",
    items=[
        inbound_api.InboundShipmentItem(
            sku="MySku1",
            quantity=36,
        )
    ]
)
```

It is up to you how you keep track of these quantity changes in your process. One way might be to cache these details in some local database. Another might be querying the current total quantity using a request to *list_inbound_shipment_items*, then calculating the new total:

```
my_shipment = "FBAMYSHIPMENT"
# Set our change quantities as "deltas", with SKU as key and the change as value
quantity_deltas = {
    'mySku1': 12,  # add 12
    'mySku2': -6,  # remove 6
}

update_items = []

list_resp = inbound_api.list_inbound_shipment_items(shipment_id=my_shipment)
for item in list_resp.parsed.ItemData.member:
    if item.SellerSKU in quantity_deltas:
        new_quantity = item.QuantityShipped + quantity_deltas[item.SellerSKU]

        # Negative quantities not permitted, so set 0 as a minimum using `max`:
        new_quantity = max([new_quantity, 0])

        # Add items to a list for updates:
```

(continues on next page)

(continued from previous page)

```
        update_items.append(
            inbound_api.InboundShipmentItem(item.SellerSKU, new_quantity)
        )

if update_items:
    update_resp = inbound_api.update_inbound_shipment(
        shipment_id=my_shipment,
        items=update_items,
    )
```

6.5.2 Adding items from a new shipment plan

Under certain conditions, items from a new shipment plan can be added to one of your existing shipments in WORKING status. In this way, you can keep a shipment "open" in your own facility, adding new items to the same shipment before "closing" it and sending it to Amazon's fulfillment network.

Follow the same steps as *Requesting a shipment plan*, then inspect the contents of the planned shipments (see *Processing shipment plans*).

Generally, you *may* be able to add newly-planned items to an existing shipment if the following details match in the target "WORKING" shipment:

- `DestinationFulfillmentCenterId`
- `LabelPrepType`
- Whether both shipments are designated for **hazmat** items.

  Note: In the author's experience, this detail may not be apparent through MWS ahead of time: you may simply need to attempt to add the item and handle whatever error occurs afterward.

  Forgiveness instead of permission, as they say.

- Whether the two shipments require **case packs** or not.

This list is not exhaustive, so use best judgment and follow Amazon's guidance where necessary.

If you determine that a planned item *can* be added to one of your existing shipments, add that item to an `update_inbound_shipment` request for the given shipment ID.

As mentioned in *Changing item quantities*, remember to use the **total** quantity of an item being updated, not the change in quantity, if the item is already present in the given shipment. If you are not tracking these quantities in your own application, you may wish to send a request to *`list_inbound_shipment_items`* to obtain the current quantity of a matching item *before* sending the update request.

CHAPTER

SEVEN

USING PARSED XML RESPONSES

New in version 1.0dev15: `MWSResponse` and `DotDict` added.

Warning: The following pertains to features added in **v1.0dev15** related to MWS requests. These features are disabled by default. To use these features, set flag `_use_feature_mwsresponse` to `True` on an API class instance *before* making any requests:

```
api_class = Orders(...)
api_class._use_feature_mwsresponse = True
```

If the flag is `False`, all requests will return either `DictWrapper` or `DataWrapper` objects (deprecated); and parsed XML contents will be returned as an instance of `ObjectDict` (deprecated).

New features using `MWSResponse` *and* `DotDict` *will become the default in v1.0.*

For most MWS operations, the returned response is an XML documents encoded using ISO 8859-1. python-amazon-mws will wrap all responses in an instance of *`MWSResponse`*, which then parses these responses automatically using the `xmltodict` package. This parsed content is then available from the *`MWSResponse.parsed`* property.

Below, we'll go into more detail on how to use `MWSResponse.parsed` in your application to get the most from these XML responses.

7.1 How XML responses are parsed in python-amazon-mws

XML responses from MWS typically look like the following example (adapted from an example in MWS documentation):

```
<?xml version="1.0"?>
<ListMatchingProductsResponse xmlns="http://mws.amazonservices.com/schema/Products/
↪2011-10-01">
  <ListMatchingProductsResult>
    <Products xmlns="http://mws.amazonservices.com/schema/Products/2011-10-01"␣
↪xmlns:ns2="http://mws.amazonservices.com/schema/Products/2011-10-01/default.xsd">
      <Product>
        <Identifiers>
          <MarketplaceASIN>
            <MarketplaceId>ATVPDKIKX0DER</MarketplaceId>
            <ASIN>059035342X</ASIN>
          </MarketplaceASIN>
        </Identifiers>
        <AttributeSets>
          <ns2:ItemAttributes xml:lang="en-US">
```

(continues on next page)

(continued from previous page)

```
            <ns2:Binding>Paperback</ns2:Binding>
            <ns2:Brand>Scholastic Press</ns2:Brand>
            <ns2:Creator Role="Illustrator">GrandPrÃ©, Mary</ns2:Creator>
          </ns2:ItemAttributes>
        </AttributeSets>
        <Relationships/>
      </Product>
    </Products>
  </ListMatchingProductsResult>
  <ResponseMetadata>
    <RequestId>3b805a12-689a-4367-ba86-EXAMPLE91c0b</RequestId>
  </ResponseMetadata>
</ListMatchingProductsResponse>
```

Parsing of this document goes through the following steps in python-amazon-mws:

1. All requests are sent through the `requests` package, and responses are returned as a `requests.Response` instance.

   The `Response` object is then wrapped by *`MWSResponse`*, and stored internally as *`MWSResponse.original`*.

2. If the response did not specify an encoding in its headers, `MWSResponse` will call on `requests.Response.apparent_encoding` explicitly to force character set detection. For most use cases, this will allow the *`MWSResponse.text`* property to decode the response content properly.

   Note: if a different encoding is required, you can alter *`MWSResponse.encoding`* before accessing `MWSResponse.text`, or work with the raw *`MWSResponse.content`*.

   You can also initialize an API class instance with a `force_response_encoding='my-encoding'` argument. This will override the encoding used to decode all responses from that API's requests. This is useful when you are confident that responses are being encoded differently, such as when responses are actually encoded in UTF-8 (despite Amazon's documentation to the contrary).

   ---

3. *`MWSResponse.parse_response()`* is called, which:

   1. Produces a "clean" copy of the XML document to use for parsing (see *XML "cleaning" before parsing*). (The original response content is left unchanged: only a copy is used for parsing.)
   2. Runs `MWSResponse.text` through the utility `mws.utils.xml.mws_xml_to_dict`. This uses `xmltodict.parse()` to convert the XML to a standard Python dictionary, which is returned and stored as `MWSResponse._dict`.
   3. Wraps the parsed Python dict in a *`DotDict`*, which can be accessed from *`MWSResponse.parsed`*.

      If the response contains a `<ResponseMetadata>` tag, this method also builds a `DotDict` instance of this key only, storing it as *`MWSResponse.metadata`*. Typically this tag only contains the `<RequestId>` element, so the property *`MWSResponse.request_id`* can also be used to access this value.

Once parsing is complete, the *`MWSResponse`* instance is returned. From this instance, we can access the *`DotDict`* that is returned from its *`.parsed`* property to comb through the returned data.

For more details on how to make the best use of this parsed data, please see *DotDict*.

7.2 Result keys and metadata

Most MWS requests returning XML documents take the following overall shape:

```
<?xml version="1.0"?>
<OperationResponse>
  <OperationResult>
    ...
  </OperationResult>
  <ResponseMetadata>
    <RequestId>...</RequestId>
  </ResponseMetadata>
</OperationResponse>
```

The parsed document initially returns a `dict` with just two keys. For the above example, that would look like so:

```
{
    'OperationResult': ...,
    'ResponseMetadata': ...,
}
```

Note: `Operation` in all above examples would be replaced by the name of the MWS operation that was called. For the `ListInboundShipments` operation, for example, the document's root will be `ListInboundShipmentsResponse`, and the result key will be `ListInboundShipmentsResult`.

Both the `...Result` key and `ResponseMetadata` are accessible from *MWSResponse* through separate properties:

- The `...Result` key is used as the root for *MWSResponse.parsed*, so accessing `.parsed` should only return parsed content found inside the `<...Result>` tag.
- `ResponseMetadata` is accessible from *MWSResponse.metadata*. You can access the `RequestId` stored there either as `MWSResponse.metadata.RequestId` or through the shortcut property, *MWSResponse.request_id*.

Tip: Amazon recommends logging `RequestId` as well as the request timestamp (found in `MWSResponse.timestamp`) to aid in troubleshooting when contacting their support channels.

7.3 XML "cleaning" before parsing

MWS XML responses may be returned with a variety of data that does not fit well into Python data structures During parsing of these responses, python-amazon-mws either removes or finesses some of this data into a "cleaner" format.

Consider the example response from earlier:

```
<?xml version="1.0"?>
<ListMatchingProductsResponse xmlns="http://mws.amazonservices.com/schema/Products/
↪2011-10-01">
  <ListMatchingProductsResult>
    <Products xmlns="http://mws.amazonservices.com/schema/Products/2011-10-01"␣
↪xmlns:ns2="http://mws.amazonservices.com/schema/Products/2011-10-01/default.xsd">
      <Product>
```

(continues on next page)

(continued from previous page)

```
6         <Identifiers>
7           <MarketplaceASIN>
8             <MarketplaceId>ATVPDKIKX0DER</MarketplaceId>
9             <ASIN>059035342X</ASIN>
10          </MarketplaceASIN>
11        </Identifiers>
12        <AttributeSets>
13          <ns2:ItemAttributes xml:lang="en-US">
14            <ns2:Binding>Paperback</ns2:Binding>
15            <ns2:Brand>Scholastic Press</ns2:Brand>
16            <ns2:Creator Role="Illustrator">GrandPrÃ©, Mary</ns2:Creator>
17          </ns2:ItemAttributes>
18        </AttributeSets>
19        <Relationships/>
20      </Product>
21    </Products>
22  </ListMatchingProductsResult>
23  <ResponseMetadata>
24    <RequestId>3b805a12-689a-4367-ba86-EXAMPLE91c0b</RequestId>
25  </ResponseMetadata>
26 </ListMatchingProductsResponse>
```

This document will be "cleaned" as follows:

- The document's root tag - in this case `<ListMatchingProductsResponse>` - will be ignored. The parsed Python dict will take the shape of:

```
{
    'ListMatchingProductsResult': ...,
    'ResponseMetadata': ...
}
```

- Namespaces are removed. For instance, the `<Products>` tag (line 4) will have both namespaces stripped, leaving only the bare tag name.
- Prefixes - such as `ns2:` or `xml:`, seen on lines 13 through 17 - are removed from tag names and attributes. The tag `<ns2:ItemAttributes xml:lang="en-US">` on line 13 will be stripped down to just `<ItemAttributes lang="en-US">` prior to being parsed.

CHAPTER

EIGHT

FEEDS

According to Amazon's documentation:

> The Amazon MWS Feeds API section of the Amazon Marketplace Web Service (Amazon MWS) API lets you upload inventory and order data to Amazon. You can also use the Amazon MWS Feeds API section to get information about the processing of feeds.

More details on how to utilize the Feeds API per MWS requirements can be found at the above link. Below we'll provide details on how to use this API with *Feeds*.

8.1 Uploading metadata for VAT invoices

Metadata for VAT invoices is processed as a `FeedOptions` parameter to the `SubmitFeed` operation, as described in Amazon's documentation, Invoice Uploader Developer Guide (PDF) This parameter is not described in the standard MWS developer documentation, unfortunately, which can lead to some confusion.

When submitting a feed, you can either build the metadata string yourself following the above guidelines, or provide a dict to the `feed_options` arg for *Feeds.submit_feed*:

```
from mws import Feeds, Marketplaces

feeds_api = Feeds(MY_ACCESS_KEY, MY_SECRET_KEY, MY_ACCOUNT_ID)

feed_opts = {'orderid': '407-XXXXXX-6760332', 'invoicenumber': 51}

response = feeds_api.submit_feed(
    feed=my_invoice_file.encode(),
    feed_type='_UPLOAD_VAT_INVOICE_',
    feed_options=feed_opts,
    marketplace_ids=Marketplaces.UK.marketplace_id,
)
```

The above will automatically convert `feed_opts` into the formatted string `'metadata:orderid=407-XXXXXX-6760332;metadata:invoicenumber=51'` when the request is sent. You can also send this same string as `feed_options`, if you wish to perform your own formatting:

```
response = feeds_api.submit_feed(
    feed=my_invoice_file.encode(),
    feed_type='_UPLOAD_VAT_INVOICE_',
    feed_options='metadata:orderid=407-XXXXXX-6760332;metadata:invoicenumber=51',
    marketplace_ids=Marketplaces.UK.marketplace_id,
)
```

Note: The format for the FeedOptions string is described in Amazon's documentation here (PDF). You are welcome to format your own FeedOptions string, if you find that the python-amazon-mws implementation is not suitable for your specific needs.

You can find our implementation for this formatting within the source for `Feeds`.

8.2 Feeds API reference

class mws.**Feeds**(*access_key*, *secret_key*, *account_id*, *region='US'*, *uri=''*, *version=''*, *auth_token=''*, *proxy=None*, *user_agent_str=''*, *headers=None*, *force_response_encoding=None*)

Amazon MWS Feeds API.

Docs: https://docs.developer.amazonservices.com/en_US/feeds/Feeds_Overview.html

submit_feed(*feed*, *feed_type*, *feed_options=None*, *marketplace_ids=None*, *amazon_order_id=None*, *document_type=None*, *content_type='text/xml'*, *purge=False*)

The SubmitFeed operation. Uploads a feed for processing by Amazon MWS.

Requires `feed`, a file in XML or flat-file format encoded to bytes; and `feed_type`, a string detailing a FeedType enumeration.

All other parameters may change depending on the `feed_type` you select. See Amazon docs for details.

`feed_options` is used for `feed_type` "_UPLOAD_VAT_INVOICE_", to provide FeedOption metadata. See Invoice Uploader Developer Guide (PDF), for details. Can accept a dict of simple key-value pairs, which will be converted to the proper string format automatically.

`marketplace_ids` accepts a list of one or more marketplace IDs where you want the feed to be applied. Can also accept a single marketplace ID as a string.

`amazon_order_id` and `document_type` are used for `feed_type` "_POST_EASYSHIP_DOCUMENTS_", used for Amazon Easy Ship orders (available only in India marketplace). Provide an Amazon Order ID as a string and the type of PDF document ("ShippingLabel", "Invoice", or "Warranty"; or *None* to get all).

`content_type` sets the "Content-Type" request header, indicating the type of file being sent. Defaults to `"text/xml"`.

`purge` enables Amazon's "purge and replace" functionality. Set to `True` to purge and replace existing data, otherwise use `False` (the default). Only applies to product-related flat file feed types. **Use only in exceptional cases.** Usage is throttled to allow only one purge and replace within a 24-hour period.

get_feed_submission_list(*feed_ids=None*, *max_count=None*, *feed_types=None*, *processing_statuses=None*, *from_date=None*, *to_date=None*, *next_token=None*)

Returns a list of all feed submissions submitted between *from_date* and *to_date*. If these params are omitted, defaults to the previous 90 days.

Pass *next_token* to call "GetFeedSubmissionListByNextToken" instead.

Docs: https://docs.developer.amazonservices.com/en_US/feeds/Feeds_GetFeedSubmissionList.html

get_feed_submission_list_by_next_token(*token*)

Alias for *get_feed_submission_list(next_token=token)*.

Docs: https://docs.developer.amazonservices.com/en_US/feeds/Feeds_GetFeedSubmissionListByNextToken.html

get_feed_submission_count(*feed_types=None*, *processing_statuses=None*, *from_date=None*, *to_date=None*)

Returns a count of the feeds submitted between *from_date* and *to_date*. If these params are omitted, defaults to the previous 90 days.

Docs: https://docs.developer.amazonservices.com/en_US/feeds/Feeds_GetFeedSubmissionCount.html

cancel_feed_submissions(*feed_ids=None*, *feed_types=None*, *from_date=None*, *to_date=None*)

Cancels one or more feed submissions and returns a count of the feed submissions that were canceled.

Docs: https://docs.developer.amazonservices.com/en_US/feeds/Feeds_CancelFeedSubmissions.html

get_feed_submission_result(*feed_id*)

Returns the feed processing report and the Content-MD5 header.

Docs: https://docs.developer.amazonservices.com/en_US/feeds/Feeds_GetFeedSubmissionResult.html

class FeedProcessingStatus(*value*)

Enumerates all the feed processing status values that are available through the Feeds API section.

MWS Docs: FeedProcessingStatus enumeration

class FeedType(*value*)

Enumerates all the feed types that are available through the Feeds API section.

MWS Docs: FeedType enumeration

Please refer to MWS documentation for details on each FeedType, including usage, template files, and additional information links.

CHAPTER

NINE

INBOUNDSHIPMENTS

According to Amazon's documentation:

> With the Fulfillment Inbound Shipment API section of Amazon Marketplace Web Service (Amazon MWS), you can create and update inbound shipments of inventory in Amazon's fulfillment network. You can also request lists of inbound shipments or inbound shipment items based on criteria that you specify. After your inventory has been received in the fulfillment network, Amazon can fulfill your orders regardless of whether you are selling on Amazon's retail web site or through other retail channels.

9.1 InboundShipments API reference

class mws.**InboundShipments**(**args*, ***kwargs*)

Amazon MWS FulfillmentInboundShipment API

MWS docs: FulfillmentInboundShipment Overview

set_ship_from_address(*address*)

DEPRECATED, remove later. Now an alias to assigning `from_address` property directly.

Parameters **address** (*Union[mws.models.inbound_shipments.Address, dict]*) –

from_address_params(*from_address=None*, *prefix=''*)

Converts a from address, either stored or passed as an argument, to params.

If provided as an argument, checks first that the arg is the correct type, raising TypeError if it's not an instance of the Address model.

Providing a from_address as an argument will override any address stored on this API instance.

Parameters

- **from_address** (*Optional[mws.models.inbound_shipments.Address]*) –
- **prefix** (*str*) –

Return type dict

get_inbound_guidance_for_sku(*skus*, *marketplace_id*)

Returns inbound guidance for a list of items by Seller SKU.

`skus` expects some iterable of strings. If it is any other type of object, it will be treated as a single instance and wrapped in a list first, similar to passing `[skus]`.

MWS docs: GetInboundGuidanceForSKU

Parameters

- **skus** (*Union[List[str], str]*) –
- **marketplace_id** (*str*) –

get_inbound_guidance_for_asin(*asins*, *marketplace_id*)
Returns inbound guidance for a list of items by ASIN.

`asins` expects some iterable of strings. If it is any other type of object, it will be treated as a single instance and wrapped in a list first, similar to passing `[asins]`.

MWS docs: GetInboundGuidanceForASIN

Parameters

- **asins** (*Union[List[str], str]*) –
- **marketplace_id** (*str*) –

create_inbound_shipment_plan(*items*, *country_code='US'*, *subdivision_code=None*, *label_preference=None*, *from_address=None*)
Returns one or more inbound shipment plans, which provide the information you need to create inbound shipments.

`items` expects a list of `InboundShipmentPlanRequestItem` model instances. Also supports a list of "legacy" dictionaries, in which the keys 'sku' and 'quantity' are required; and keys 'asin', 'condition', and 'quantity_in_case' are optional.

- Note that the dictionary format does not support adding `PrepDetails`, as the `InboundShipmentPlanRequestItem` model does.

If `from_address` is not provided (with an instance of the `Address` model), then the `.from_address` attribute of this class instance must be set before using this operation.

MWS docs: CreateInboundShipmentPlan

Parameters

- **items** (*List[Union[mws.models.inbound_shipments.InboundShipmentPlanRequestItem, dict]]*) –
- **country_code** (*str*) –
- **subdivision_code** (*Optional[str]*) –
- **label_preference** (*Optional[str]*) –
- **from_address** (*Optional[mws.models.inbound_shipments.Address]*) –

create_inbound_shipment(*shipment_id*, *shipment_name*, *destination*, *items*, *shipment_status='WORKING'*, *label_preference=None*, *case_required=False*, *box_contents_source=None*, *from_address=None*)
Creates an inbound shipment to Amazon's fulfillment network.

`items` expects a list of `InboundShipmentItem` model instances. Also supports a list of "legacy" dictionaries, in which the keys 'sku' and 'quantity' are required; and key 'quantity_in_case' is optional.

- Note that the dictionary format does not support adding `PrepDetails`, as the `InboundShipmentItem` model does.
- The model also supports adding `release_date`, which the dictionary does not.

If `from_address` is not provided (with an instance of the `Address` model), then the `.from_address` attribute of this class instance must be set before using this operation.

MWS docs: CreateInboundShipment

Parameters

- **shipment_id** (*str*) –
- **shipment_name** (*str*) –
- **destination** (*str*) –
- **items** (*List[Union[mws.models.inbound_shipments.InboundShipmentItem, dict]]*) –
- **shipment_status** (*str*) –
- **label_preference** (*Optional[str]*) –
- **case_required** (*bool*) –
- **box_contents_source** (*Optional[str]*) –
- **from_address** (*Optional[mws.models.inbound_shipments.Address]*) –

update_inbound_shipment(*shipment_id*, *shipment_name=None*, *destination=None*, *items=None*, *shipment_status=None*, *label_preference=None*, *case_required=None*, *box_contents_source=None*, *from_address=None*)

Updates an existing inbound shipment in Amazon FBA.

`items` expects a list of `InboundShipmentItem` model instances. Also supports a list of "legacy" dictionaries, in which the keys 'sku' and 'quantity' are required; and key 'quantity_in_case' is optional.

- Note that the dictionary format does not support adding `PrepDetails`, as the `InboundShipmentItem` model does.
- The model also supports adding `release_date`, which the dictionary does not.

If `from_address` is not provided (with an instance of the `Address` model), then the `.from_address` attribute of this class instance must be set before using this operation.

MWS docs: UpdateInboundShipment

Parameters

- **shipment_id** (*str*) –
- **shipment_name** (*Optional[str]*) –
- **destination** (*Optional[str]*) –
- **items** (*Optional[List[Union[mws.models.inbound_shipments.InboundShipmentItem, dict]]]*) –
- **shipment_status** (*Optional[str]*) –
- **label_preference** (*Optional[str]*) –
- **case_required** (*Optional[bool]*) –
- **box_contents_source** (*Optional[str]*) –
- **from_address** (*Optional[mws.models.inbound_shipments.Address]*) –

get_preorder_info(*shipment_id*)

Returns pre-order information, including dates, that a seller needs before confirming a shipment for pre-order. Also indicates if a shipment has already been confirmed for pre-order.

MWS docs: GetPreorderInfo

Parameters **shipment_id** (*str*) –

confirm_preorder(*shipment_id*, *need_by_date*)
Confirms a shipment for pre-order.

MWS docs: ConfirmPreorder

Parameters

- **shipment_id** (*str*) –
- **need_by_date** (*datetime.datetime*) –

get_prep_instructions_for_sku(*skus*, *country_code='US'*)
Returns labeling requirements and item preparation instructions to help you prepare items for an inbound shipment.

MWS docs: GetPrepInstructionsForSKU

Parameters

- **skus** (*Union[List[str], str]*) –
- **country_code** (*str*) –

get_prep_instructions_for_asin(*asins*, *country_code='US'*)
Returns item preparation instructions to help with item sourcing decisions.

MWS docs: GetPrepInstructionsForASIN

Parameters

- **asins** (*Union[List[str], str]*) –
- **country_code** (*str*) –

estimate_transport_request(*shipment_id*)
Requests an estimate of the shipping cost for an inbound shipment.

MWS docs: EstimateTransportRequest

Parameters **shipment_id** (*str*) –

get_transport_content(*shipment_id*)
Returns current transportation information about an inbound shipment.

MWS docs: GetTransportContent

Parameters **shipment_id** (*str*) –

confirm_transport_request(*shipment_id*)
Confirms that you accept the Amazon-partnered shipping estimate and you request that the Amazon-partnered carrier ship your inbound shipment.

MWS docs: ConfirmTransportRequest

Parameters **shipment_id** (*str*) –

void_transport_request(*shipment_id*)
Voids a previously-confirmed request to ship your inbound shipment using an Amazon-partnered carrier.

MWS docs: VoidTransportRequest

Parameters **shipment_id** (*str*) –

get_package_labels(*shipment_id*, *num_labels*, *page_type=None*)
Returns PDF document data for printing package labels for an inbound shipment.

MWS docs: GetPackageLabels

Parameters

- **shipment_id** (*str*) –
- **num_labels** (*int*) –
- **page_type** (*str*) –

get_unique_package_labels(*shipment_id*, *page_type*, *package_ids*)
Returns unique package labels for faster and more accurate shipment processing at the Amazon fulfillment center.

`shipment_id` must match a valid, current shipment.

`page_type` expected to be string matching one of following (not checked, in case Amazon requirements change):

- "PackageLabel_Letter_2"
- "PackageLabel_Letter_6"
- "PackageLabel_A4_2"
- "PackageLabel_A4_4"
- "PackageLabel_Plain_Paper"

`package_ids` expects some iterable of strings or integers. If it is any other type of object, it will be treated as a single instance and wrapped in a list first, similar to passing `[package_ids]`.

MWS docs: GetUniquePackageLabels

Parameters

- **shipment_id** (*str*) –
- **page_type** (*str*) –
- **package_ids** (*Union[Iterable[Union[str, int]], str, int]*) –

get_pallet_labels(*shipment_id*, *page_type*, *num_labels*)
Returns `num_labels` number of pallet labels for shipment `shipment_id` of the given `page_type`.

Amazon expects `page_type` as a string matching one of following:

- "PackageLabel_Letter_2"
- "PackageLabel_Letter_6"
- "PackageLabel_A4_2"
- "PackageLabel_A4_4"
- "PackageLabel_Plain_Paper"

`num_labels` is integer, number of labels to create.

MWS docs: GetPalletLabels

Parameters

- **shipment_id** (*str*) –
- **page_type** (*str*) –

- **num_labels** (*int*) –

get_bill_of_lading(*shipment_id*)
Returns PDF document data for printing a bill of lading for an inbound shipment.

MWS docs: GetBillOfLading

Parameters **shipment_id** (*str*) –

list_inbound_shipments(*shipment_ids=None*, *shipment_statuses=None*, *last_updated_after=None*, *last_updated_before=None*, *next_token=None*)
Returns list of shipments based on statuses, IDs, and/or before/after datetimes.

Pass `next_token` to call "ListInboundShipmentsByNextToken" instead.

MWS docs: ListInboundShipments

Parameters

- **shipment_ids** (*Iterable[str]*) –
- **shipment_statuses** (*Iterable[str]*) –
- **last_updated_after** (*datetime.datetime*) –
- **last_updated_before** (*datetime.datetime*) –
- **next_token** (*str*) –

list_inbound_shipments_by_next_token(*token*)
Alias for `list_inbound_shipments(next_token=token)`

MWS docs: ListInboundShipmentsByNextToken

Parameters **token** (*str*) –

list_inbound_shipment_items(*shipment_id=None*, *last_updated_after=None*, *last_updated_before=None*, *next_token=None*)
Returns list of items within inbound shipments and/or before/after datetimes.

Pass `next_token` to call "ListInboundShipmentItemsByNextToken" instead.

MWS docs: ListInboundShipmentItems

Parameters

- **shipment_id** (*str*) –
- **last_updated_after** (*datetime.datetime*) –
- **last_updated_before** (*datetime.datetime*) –
- **next_token** (*str*) –

list_inbound_shipment_items_by_next_token(*token*)
Alias for `list_inbound_shipment_items(next_token=token)`

MWS docs: ListInboundShipmentItemsByNextToken

Parameters **token** (*str*) –

9.2 Other tools

Note: The following classes and utility functions are attached to the `InboundShipments` class for convenient access. For example, the *`Address`* model can be accessed like so:

```
from mws import InboundShipments

my_address = InboundShipments.Address(...)

# or from an instance of InboundShipments:

inbound_api = InboundShipments(...)
my_address = inbound_api.Address(...)
```

9.2.1 Data models

class mws.models.inbound_shipments.**Address**(*name=None*, *address_line_1=None*, *address_line_2=None*, *city=None*, *district_or_county=None*, *state_or_province_code=None*, *country_code='US'*, *postal_code=None*)

Postal address information.

MWS docs: Address Datatype

classmethod from_legacy_dict(*value*)

Create an Address from a legacy structured dict.

Parameters **value** (*dict*) –

Return type *mws.models.inbound_shipments.Address*

to_params(*prefix=''*)

Flattens all parameters and values of this model into a single key-value dictionary, suitable for use in a request to MWS.

Parameters **prefix** (*str*) –

Return type dict

class mws.models.inbound_shipments.**PrepDetails**(*prep_instruction*, *prep_owner='SELLER'*)

A preparation instruction, and who is responsible for that preparation.

MWS docs: PrepDetails Datatype

`prep_instruction` accepts either a string or an instance of the `PrepInstruction` enum, detailing the type of prep to perform.

`prep_owner` (optional) accepts a string, typically "AMAZON" or "SELLER", to indicate who is responsible for the prep. You can use `PrepDetails.AMAZON` or `PrepDetails.SELLER` to fill in these values. Defaults to "SELLER".

to_params(*prefix=''*)

Flattens all parameters and values of this model into a single key-value dictionary, suitable for use in a request to MWS.

Parameters **prefix** (*str*) –

Return type dict

class mws.models.inbound_shipments.**InboundShipmentPlanRequestItem**(**args*, *asin=None*, *condition=None*, ***kwargs*)

Item information for creating an inbound shipment plan. Submitted with a call to the CreateInboundShipmentPlan operation.

MWS docs: InboundShipmentPlanRequestItem Datatype

Adds the optional arguments `asin` (to include ASIN as needed) and `condition` (to add item condition information).

`condition` may be a string or an instance of `ItemCondition`.

to_params(*prefix=''*)

Flattens all parameters and values of this model into a single key-value dictionary, suitable for use in a request to MWS.

Parameters **prefix** (*str*) –

Return type dict

class mws.models.inbound_shipments.**InboundShipmentItem**(**args*, *release_date=None*, ***kwargs*)

Item information for an inbound shipment. Submitted with a call to the CreateInboundShipment or UpdateInboundShipment operation.

MWS docs: InboundShipmentItem Datatype

classmethod from_plan_item(*item*, *quantity_in_case=None*, *release_date=None*)

Construct this model from a shipment plan returned from a CreateInboundShipmentPlan request.

Expects a `DotDict` instance that can typically be found in the parsed response object by:

1. Iterating `for plan in resp.parsed.InboundShipmentPlans.member:`; and
2. Iterating `for item in plan.Items.member:`.

Each `item` instance in the above example *should* work here [YMMV].

`quantity_in_case` must be passed manually for case-packed shipments, even when constructing from a shipment plan response, as this data is not typically returned in the plan details.

`release_date` is also not part of a shipment plan response, so this must be passed manually in order to add it to the item.

Parameters

- **item** (*mws.utils.collections.DotDict*) –
- **quantity_in_case** (*Optional[int]*) –
- **release_date** (*Optional[datetime.datetime]*) –

Return type *mws.models.inbound_shipments.InboundShipmentItem*

to_params(*prefix=''*)

Flattens all parameters and values of this model into a single key-value dictionary, suitable for use in a request to MWS.

Parameters **prefix** (*str*) –

Return type dict

9.2.2 Enums

class mws.models.inbound_shipments.**PrepInstruction**(*value*)
Bases: enum.Enum

Enumeration of preparation instruction types.

MWS docs: PrepInstruction Datatype

POLYBAGGING = 'Polybagging'

BUBBLEWRAPPING = 'BubbleWrapping'

TAPING = 'Taping'

BLACKSHRINKWRAPPING = 'BlackShrinkWrapping'

LABELING = 'Labeling'

HANGGARMENT = 'HangGarment'

class mws.models.inbound_shipments.**ItemCondition**(*value*)
Bases: str, enum.Enum

Condition value for an item included with a CreateInboundShipmentPlan request. Values are defined within the InboundShipmentPlanRequestItem Datatype documentation.

NEW_ITEM = 'NewItem'

NEW_WITH_WARRANTY = 'NewWithWarranty'

NEW_OEM = 'NewOEM'

NEW_OPEN_BOX = 'NewOpenBox'

USED_LIKE_NEW = 'UsedLikeNew'

USED_VERY_GOOD = 'UsedVeryGood'

USED_GOOD = 'UsedGood'

USED_ACCEPTABLE = 'UsedAcceptable'

USED_POOR = 'UsedPoor'

USED_REFURBISHED = 'UsedRefurbished'

COLLECTIBLE_LIKE_NEW = 'CollectibleLikeNew'

COLLECTIBLE_VERY_GOOD = 'CollectibleVeryGood'

COLLECTIBLE_GOOD = 'CollectibleGood'

COLLECTIBLE_ACCEPTABLE = 'CollectibleAcceptable'

COLLECTIBLE_POOR = 'CollectiblePoor'

REFURBISHED_WITH_WARRANTY = 'RefurbishedWithWarranty'

REFURBISHED = 'Refurbished'

CLUB = 'Club'

9.2.3 Utilities

mws.models.inbound_shipments.**shipment_items_from_plan**(*plan*, *overrides=None*)

Given a shipment plan response, returns a list of `InboundShipmentItem` models constructed from the contents of that plan's `Items` set.

Expects `plan` to be a node from a parsed MWS response from the `create_inbound_shipment_plan` request, typically the `resp.parsed.InboundShipmentPlans.member` node (which may be a *DotDict* for a single plan or a list of `DotDict` instances for multiple; though both options should be natively iterable with the same interface).

Providing `overrides` allows the addition of details that are not returned by `create_inbound_shipment_plan`, such as `quantity_in_case` and `release_date`. Expects a dict where SellerSKUs are keys and the values are either instances of `ExtraItemData` or dictionaries with the keys `quantity_in_case` and/or `release_date`. Only items matching a SellerSKU key in `overrides` will have data overridden this way.

For example usage, see: *Converting plan items to shipment items*

Parameters

- **plan** (*Union[mws.utils.collections.DotDict, List[mws.utils.collections.DotDict]]*) –
- **overrides** (*Optional[Dict[str, mws.models.inbound_shipments.ExtraItemData]]*) –

Return type List[*mws.models.inbound_shipments.InboundShipmentItem*]

class mws.models.inbound_shipments.**ExtraItemData**(*quantity_in_case=None*, *release_date=None*)

Dataclass used for providing overrides to individual SKUs when processing items from a planned shipment in bulk using *shipment_items_from_plan()*.

To utilize this data, construct a dictionary that maps SellerSKUs to instances of this class, then pass that dictionary to the `overrides` argument for `shipment_items_from_plan`.

Example:

```
override_data = {
    # with a case quantity
    "MySku1": ExtraItemData(quantity_in_case=12),
    # a release date
    "MySku2": ExtraItemData(release_date=datetime.datetime(2021, 1, 28)),
    # or both (short version)
    "MySku3": ExtraItemData(24, datetime.datetime(2021, 1, 28)),
}

data = shipment_items_from_plan(plan, override_data)
```

CHAPTER

TEN

PRODUCTS

According to Amazon's documentation:

> The Products API section of Amazon Marketplace Web Service (Amazon MWS) helps you get information to match your products to existing product listings on Amazon Marketplace websites and to make sourcing and pricing decisions for listing those products on Amazon Marketplace websites. The Amazon MWS Products API returns product attributes, current Marketplace pricing information, and a variety of other product and listing information.

10.1 Using examples on this page

All examples below assume you have setup your Products API instance appropriately. Refer to *Getting started* for details:

```
from mws import Products

products_api = Products(
    access_key="...",
    secret_key="...",
    account_id="...",
    auth_token="...",
)
```

All request methods in the Products API also require a **MarketplaceId** to specify which marketplace the products are sold in. MarketplaceId values should match one of the values specified in Amazon documentation: Amazon MWS endpoints and MarkeplaceId values

python-amazon-mws makes these values available through the `Marketplaces` Enum, which contains both the `endpoint` and `marketplace_id` for each Amazon region via that region's country code.

For convenience, a `Marketplaces` instance will return its MarketplaceId through the `.value` attribute, as well. Further, all request methods in python-amazon-mws will automatically "clean" Enum instances by returning their `.value` attributes.

The following are all valid methods for obtaining, for example, the MarketplaceId for the US region and passing it to a request method in the `Products` API:

```
from mws import Marketplaces

my_market = Marketplaces.US
# Returns the Enum instance for the US region.
# When used in a request method, the `marketplace_id` value will be used␣
↪automatically.
```

(continues on next page)

(continued from previous page)

```
print(my_market.marketplace_id)
# 'ATVPDKIKX0DER'
print(my_market.value)
# 'ATVPDKIKX0DER'
# (alias for `.marketplace_id`)

# You can also return the endpoint for that region, if needed:
print(my_market.endpoint)
# 'https://mws.amazonservices.com'
```

In all examples below, replace `my_market` with the `Marketplaces` Enum instance or MarketplaceId string value relevant to your region.

10.2 Products API reference

class mws.**Products**(*access_key*, *secret_key*, *account_id*, *region='US'*, *uri=''*, *version=''*, *auth_token=''*, *proxy=None*, *user_agent_str=''*, *headers=None*, *force_response_encoding=None*)

Amazon MWS Products API

MWS Docs: Products API Overview

list_matching_products(*marketplace_id*, *query*, *context_id=None*)

Returns a list of products and their attributes, based on a search query.

MWS Docs: ListMatchingProducts

Examples:

- Obtaining ASINs for products returned by the query `"Python"`:

```
resp = products_api.list_matching_products(
    marketplace_id=my_market,
    query="Python",
)

for product in resp.parsed.Products.Product:
    asin = product.Identifiers.MarketplaceASIN.ASIN
    print(f"ASIN: {asin}")
```

Note: As a shorthand, you may access the first product from the response using a list index:

```
resp.parsed.Products.Product[0].Identifiers.MarketplaceASIN.ASIN
```

Beware: if only one product is returned, this may result in an error, as the `Product` node will not be a list. Iterating nodes is generally safer to avoid this issue (see: *DotDict Native Iteration*).

- Returning sales rank categories and rank numbers:

```
for product in resp.parsed.Products.Product:
    for rank in product.SalesRankings.SalesRank:
        category_id = rank.ProductCategoryId
```

(continues on next page)

(continued from previous page)

```
        sales_rank = rank.Rank
        print(f"Category: {category_id}, Rank: {sales_rank}")
```

- Returning product titles:

```
for product in resp.parsed.Products.Product:
    product_title = product.AttributeSets.ItemAttributes.Title
    print(f"Title: {product_title}")
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **query** (*str*) –
- **context_id** (*Optional[str]*) –

get_matching_product (*marketplace_id*, *asins*)

Returns a list of products and their attributes, based on a list of ASIN values.

MWS Docs: GetMatchingProduct

Example

```
resp = products_api.get_matching_product(
    marketplace_id=my_market,
    asins=["B085G58KWT", "B07ZZW7QCM"],
)

# Iterate over products returned by the request
for product in resp.parsed.Product:
    # Access identifiers
    print(product.Identifiers.MarketplaceASIN.ASIN)
    print(product.Identifiers.MarketplaceASIN.MarketplaceId)

    # Attributes of the product, for instance a ListPrice (by amount and␣
↪currency code):
    print(product.AttributeSets.ItemAttributes.ListPrice.Amount)
    print(product.AttributeSets.ItemAttributes.ListPrice.CurrencyCode)
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **asins** (*Union[List[str], str]*) –

get_matching_product_for_id (*marketplace_id: str*, *type_: str*, *ids: Union[List[str], str]*)

Returns a list of products and their attributes, based on a list of ASIN, GCID, SellerSKU, UPC, EAN, ISBN, and JAN values.

MWS Docs: GetMatchingProductForId

Example

```
resp = products_api.get_matching_product_for_id(
    marketplace_id=my_market,
    type_="ASIN",
    ids=["B085G58KWT", "B07ZZW7QCM"],
)
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **type_** (*str*) –
- **ids** (*Union[List[str], str]*) –

get_competitive_pricing_for_sku(*marketplace_id*, *skus*)

Returns the current competitive price of a product, based on SellerSKU.

MWS Docs: GetCompetitivePricingForSKU

Example

```
resp = products_api.get_competitive_pricing_for_sku(
    marketplace_id=my_market,
    skus=["OO-NL0F-795Z"],
)

for product in resp.parsed.Product:
    product.CompetitivePricing.NumberOfOfferListings
    product.CompetitivePricing.CompetitivePrices.CompetitivePrice.Price.
↪LandedPrice.Amount
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **skus** (*Union[List[str], str]*) –

get_competitive_pricing_for_asin(*marketplace_id*, *asins*)

Returns the current competitive price of a product, based on ASIN.

MWS Docs: GetCompetitivePricingForASIN

Example

```
resp = products_api.get_competitive_pricing_for_asin(
    marketplace_id=my_market,
    asins=["B085G58KWT"],
)
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **asins** (*Union[List[str], str]*) –

get_lowest_offer_listings_for_sku(*marketplace_id*, *skus*, *condition='Any'*, *exclude_me=False*)

Returns pricing information for the lowest-price active offer listings for up to 20 products, based on SellerSKU.

MWS Docs: GetLowestOfferListingsForSKU

Example

```
resp = products_api.get_lowest_offer_listings_for_sku(
    marketplace_id=my_market,
      skus=["OO-NL0F-795Z"],
      condition="New" # Any, New, Used, Collectible, Refurbished, Club.␣
↪Default = Any
  )
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **skus** (*Union[List[str], str]*) –
- **condition** (*str*) –
- **exclude_me** (*bool*) –

get_lowest_offer_listings_for_asin(*marketplace_id*, *asins*, *condition='Any'*, *exclude_me=False*)

Returns pricing information for the lowest-price active offer listings for up to 20 products, based on ASIN.

MWS Docs: GetLowestOfferListingsForASIN

Example

```
resp = products_api.get_lowest_offer_listings_for_asin(
    marketplace_id=my_market,
    asins=["B085G58KWT"],
    condition="New" # Any, New, Used, Collectible, Refurbished, Club. Default␣
↪= Any
)
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **asins** (*Union[List[str], str]*) –
- **condition** (*str*) –
- **exclude_me** (*bool*) –

get_lowest_priced_offers_for_sku(*marketplace_id*, *sku*, *condition='New'*, *exclude_me=False*)

Returns lowest priced offers for a single product, based on SellerSKU.

MWS Docs: GetLowestPricedOffersForSKU

Example

```
resp = products_api.get_lowest_priced_offers_for_sku(
    marketplace_id=my_market,
    skus=["OO-NL0F-795Z"],
    condition="New" # Any, New, Used, Collectible, Refurbished, Club. Default
↪= Any
)
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **sku** (*str*) –
- **condition** (*str*) –
- **exclude_me** (*bool*) –

get_lowest_priced_offers_for_asin(*marketplace_id*, *asin*, *condition='New'*, *exclude_me=False*)

Returns lowest priced offers for a single product, based on ASIN.

MWS Docs: GetLowestPricedOffersForASIN

Example

```
resp = products_api.get_lowest_priced_offers_for_asin(
    marketplace_id=my_market,
    asins=["B085G58KWT"],
    condition="New" # Any, New, Used, Collectible, Refurbished, Club. Default
↪= Any
)
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **asin** (*str*) –
- **condition** (*str*) –
- **exclude_me** (*bool*) –

get_my_fees_estimate(*fees_estimate*, **fees_estimates*)

Returns the estimated fees for a list of products.

MWS Docs: GetMyFeesEstimate

Accepts one or more *FeesEstimateRequest* instances as arguments:

Example

```
estimate_request = FeesEstimateRequest(...)
resp = products_api.get_my_fees_estimate(estimate_request)
```

Multiple estimates can be requested at the same time, as well:

```
estimate_request1 = FeesEstimateRequest(...)
estimate_request2 = FeesEstimateRequest(...)
resp = products_api.get_my_fees_estimate(estimate_request1, estimate_request2,
↪ ...)
```

Parameters

- **fees_estimate** (*mws.models.products.FeesEstimateRequest*) –
- **fees_estimates** (*mws.models.products.FeesEstimateRequest*) –

get_my_price_for_sku(*marketplace_id*, *skus*, *condition=None*)

Returns pricing information for your own offer listings, based on SellerSKU.

MWS Docs: GetMyPriceForSKU

Example

```
  resp = products_api.get_my_price_for_sku(
    marketplace_id = my_market,
    skus="OO-NL0F-795Z",
    condition="New"
    # Any, New, Used, Collectible, Refurbished, Club. Default = All
)
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **skus** (*Union[List[str], str]*) –
- **condition** (*Optional[str]*) –

get_my_price_for_asin(*marketplace_id*, *asins*, *condition=None*)

Returns pricing information for your own offer listings, based on ASIN.

MWS Docs: GetMyPriceForASIN

Example

```
resp = products_api.get_my_price_for_asin(
    marketplace_id=my_market,
    asins="B07QR73T66",
    condition="New"
    # Any, New, Used, Collectible, Refurbished, Club. Default = All
)
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **asins** (*Union[List[str], str]*) –
- **condition** (*Optional[str]*) –

get_product_categories_for_sku(*marketplace_id*, *sku*)

Returns the parent product categories that a product belongs to, based on SellerSKU.

MWS Docs: GetProductCategoriesForSKU

Example

```
resp = products_api.get_product_categories_for_sku(
    marketplace_id=my_market,
    sku="OO-NL0F-795Z"
)
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **sku** (*str*) –

get_product_categories_for_asin(*marketplace_id*, *asin*)

Returns the parent product categories that a product belongs to, based on ASIN.

MWS Docs: GetProductCategoriesForASIN

Example

```
resp = products_api.get_product_categories_for_asin(
    marketplace_id=my_market,
    asin="B07QR73T66"
)
```

Parameters

- **marketplace_id** (*Union[mws.mws.Marketplaces, str]*) –
- **asin** (*str*) –

10.3 Data models

Several data models are attached to the `Products` API class, either from the class itself or an instance of it. These can be used as arguments for certain requests.

class mws.Products.**FeesEstimateRequest**(*marketplace_id*, *id_type*, *id_value*, *price_to_estimate_fees*, *is_amazon_fulfilled*, *identifier*)

A product, marketplace, and proposed price used to request estimated fees.

MWS Docs: FeesEstimateRequest

Instances of this model are required for the argument(s) of `get_my_fees_estimate`. Constructing an instance of this model requires the use of other data models in the Products API, as well.

Example

Note: In examples below, we use the `Products` class definition to locate our models:

```
from mws import Products
Products.MoneyType(...)
```

You can also access the same models from any instance of the `Products` class:

```
products_api = Products(...)
products_api.MoneyType(...)
```

1. Start by creating `MoneyType` instances to account for different prices associated with the request, such as `listing_price` and `shipping`:

```
my_price = Products.MoneyType(
    amount=123.45,
    currency_code=Products.CurrencyCode.GBP,
)
# Note the `currency_code` argument also accepts string literals of the␣
↪currency code:
my_shipping = Products.MoneyType(amount=5.00, currency_code='GBP')
```

2. Combine these prices into a `PriceToEstimateFees` instance:

```
my_product_price = Products.PriceToEstimateFees(
    listing_price=my_price,
    shipping=my_shipping,
)
```

   For the JP market only, this price to estimate fees may optionally include `Points`.

3. Use the `PriceToEstimateFees` instance along with other data to construct the final `FeesEstimateRequest` instance:

```
estimate_request = Products.FeesEstimateRequest(
    marketplace_id=my_market,
    id_type="ASIN",  # either 'ASIN' or 'SKU', indicating the type of the `id_
↪value` argument:
    id_value="B07QR73T66",
    price_to_estimate_fees=my_product_price,  # your `PriceToEstimateFees`␣
↪instance
    is_amazon_fulfilled=False,
    identifier="request001",  # a unique identifier of your choosing
)
```

class mws.Products.**PriceToEstimateFees**(*listing_price*, *shipping*, *points=None*)

Price information for a product, used to estimate fees.

MWS Docs: PriceToEstimateFees

Accepts instances of `MoneyType` for its `listing_price` and `shipping`, and optionally accepts a `Points` instance to denote a points value (in JP region only).

class mws.Products.**MoneyType**(*amount*, *currency_code*)

An amount of money in a specified currency.

MWS Docs: MoneyType

Example

```
my_money = Products.MoneyType(
    amount=3.50,
    currency_code=Products.CurrencyCode.USD,
)
```

class mws.Products.**Points**(*points_number*, *monetary_value*)

The number of Amazon Points offered with the purchase of an item. The Amazon Points program is only available in Japan.

MWS Docs: Points

Points are expressed in terms of a `points_number` and a `monetary_value` for those points, the latter of which must be an instance of `MoneyType`.

Example:

```
# A monetary value of 2000 Japanese yen
monetary_value = Products.MoneyType(
    amount=2000.0,
    currency_code=Products.CurrencyCode.JPY,
)

# Now assign the points like so:
points = Products.Points(
    points_number=35,
    monetary_value=monetary_value,
)
```

When used in a request, *points* will be converted to a set of parameters like so:

```
print(points.to_params())
# {'PointsNumber': 35, 'PointsMonetaryValue.Amount': 2000.0, 'PointsMonetaryValue.
↪CurrencyCode': <CurrencyCode.JPY: ('JPY', 'Japanese yen')>}
```

Note: You will see the `PointsMonetaryValue.CurrencyCode` element remains an instance of Enum at this stage. When used in a request, it is automatically "cleaned" to its parameterized value, `'JPY'`.

Passing the string literal `'JPY'` as the `MoneyType.currency_code` argument is also accepted.

10.4 Enums

Related Enums are also attached to the `Products` API class, and can be accessed the same way as *Data models*.

class mws.Products.**CurrencyCode**(*value*)

Bases: `str`, `enum.Enum`

Constants for currency codes supported by Amazon.

Example:

```
# 10 US dollars
listing_price = Products.MoneyType(
    amount=10.0,
    currency_code=Products.CurrencyCode.USD,
)
print(listing_price.to_params())
# {"Amount": 10.0, "CurrencyCode": "USD"}

# 30 Chinese yuan
shipping = Products.MoneyType(30.0, Products.CurrencyCode.RMB)
print(shipping.to_params())
# {"Amount": 30.0, "CurrencyCode": "RMB"}
```

USD = 'USD'
United States dollar

EUR = 'EUR'
European euro

GBP = 'GBP'
Great Britain pounds

RMB = 'RMB'
Chinese yuan

INR = 'INR'
Indian rupee

JPY = 'JPY'
Japanese yen

CAD = 'CAD'
Canadian dollar

MXN = 'MXN'
Mexican peso

CHAPTER

ELEVEN

REPORTS

According to Amazon's documentation:

> The Reports API section of the Amazon Marketplace Web Service (Amazon MWS) API lets you request various reports that help you manage your Sell on Amazon business. Report types are specified using the ReportTypes enumeration.

11.1 Reports API reference

class mws.**Reports**(*access_key*, *secret_key*, *account_id*, *region='US'*, *uri=''*, *version=''*, *auth_token=''*, *proxy=None*, *user_agent_str=''*, *headers=None*, *force_response_encoding=None*)

Amazon MWS Reports API.

MWS Docs: Reports API Overview

request_report(*report_type*, *start_date=None*, *end_date=None*, *marketplace_ids=None*, *report_options=None*)

Creates a report request and submits the request to Amazon MWS.

MWS Docs: RequestReport

Parameters

- **report_type** (*Union[mws.models.reports.ReportType, str]*) –
- **start_date** (*Union[datetime.datetime, datetime.date]*) –
- **end_date** (*Union[datetime.datetime, datetime.date]*) –
- **marketplace_ids** (*List[Union[mws.mws.Marketplaces, str]]*) –
- **report_options** (*dict*) –

get_report_request_list(*request_ids=None*, *report_types=None*, *processing_statuses=None*, *max_count=None*, *from_date=None*, *to_date=None*, *next_token=None*)

Returns a list of report requests that you can use to get the ReportRequestId for a report.

Pass `next_token` with no other arguments to call the **GetReportRequestListByNextToken** operation, requesting the next page of results.

MWS Docs: GetReportRequestList

Parameters

- **request_ids** (*List[str]*) –
- **report_types** (*List[Union[mws.models.reports.ReportType, str]]*) –

- **processing_statuses** (*List[Union[mws.models.reports.ProcessingStatus, str]]*) –
- **max_count** (*int*) –
- **from_date** (*Union[datetime.datetime, datetime.date]*) –
- **to_date** (*Union[datetime.datetime, datetime.date]*) –
- **next_token** (*str*) –

get_report_request_list_by_next_token(*token*)

Alias for `get_report_request_list(next_token=token)`.

MWS Docs: GetReportRequestListByNextToken

Parameters **token** (*str*) –

get_report_request_count(*report_types=None*, *processing_statuses=None*, *from_date=None*, *to_date=None*)

Returns a count of report requests that have been submitted to Amazon MWS for processing.

MWS Docs: GetReportRequestCount

Parameters

- **report_types** (*List[Union[mws.models.reports.ReportType, str]]*) –
- **processing_statuses** (*List[Union[mws.models.reports.ProcessingStatus, str]]*) –
- **from_date** (*Union[datetime.datetime, datetime.date]*) –
- **to_date** (*Union[datetime.datetime, datetime.date]*) –

cancel_report_requests(*request_ids=None*, *report_types=None*, *processing_statuses=None*, *from_date=None*, *to_date=None*)

Cancels one or more report requests.

MWS Docs: CancelReportRequests

Parameters

- **request_ids** (*Optional[List[str]]*) –
- **report_types** (*Optional[List[Union[mws.models.reports.ReportType, str]]]*) –
- **processing_statuses** (*Optional[List[Union[mws.models.reports.ProcessingStatus, str]]]*) –
- **from_date** (*Optional[Union[datetime.datetime, datetime.date]]*) –
- **to_date** (*Optional[Union[datetime.datetime, datetime.date]]*) –

get_report_list(*request_ids=None*, *max_count=None*, *report_types=None*, *acknowledged=None*, *from_date=None*, *to_date=None*, *next_token=None*)

Returns a list of reports that were created between fromdate and todate (defaults to previous 90 days if ommitted).

Pass `next_token` with no other arguments to call the **GetReportListByNextToken** operation, requesting the next page of results.

MWS Docs: GetReportList

Parameters

- **request_ids** (*List[str]*) –
- **max_count** (*int*) –
- **report_types** (*List[Union[mws.models.reports.ReportType, str]]*) –
- **acknowledged** (*bool*) –
- **from_date** (*Union[datetime.datetime, datetime.date]*) –
- **to_date** (*Union[datetime.datetime, datetime.date]*) –
- **next_token** (*str*) –

get_report_list_by_next_token(*token*)

Alias for `get_report_list(next_token=token)`.

MWS Docs: GetReportListByNextToken

Parameters **token** (*str*) –

get_report_count(*report_types=None*, *acknowledged=None*, *from_date=None*, *to_date=None*)

Returns a count of the reports, created in the previous 90 days, with a status of _DONE_ and that are available for download.

MWS Docs: GetReportCount

Parameters

- **report_types** (*List[Union[mws.models.reports.ReportType, str]]*) –
- **acknowledged** (*bool*) –
- **from_date** (*Union[datetime.datetime, datetime.date]*) –
- **to_date** (*Union[datetime.datetime, datetime.date]*) –

get_report(*report_id*)

Returns the contents of a report and the Content-MD5 header for the returned report body.

MWS Docs: GetReport

Parameters **report_id** (*str*) –

manage_report_schedule(*report_type*, *schedule*, *schedule_date=None*)

Creates, updates, or deletes a report request schedule for a specified report type.

MWS Docs: ManageReportSchedule

Parameters

- **report_type** (*mws.models.reports.ReportType*) –
- **schedule** (*mws.models.reports.Schedule*) –
- **schedule_date** (*Optional[Union[datetime.datetime, datetime.date]]*) –

get_report_schedule_list(*report_types=None*, *next_token=None*)

Returns a list of order report requests that are scheduled to be submitted to Amazon MWS for processing.

Pass `next_token` with no other arguments to call the **GetReportScheduleListByNextToken** operation, requesting the next page of results.

MWS Docs: GetReportScheduleList

Parameters

- **report_types** (*List[Union[mws.models.reports.ReportType, str]]*) –
- **next_token** (*str*) –

get_report_schedule_list_by_next_token(*token*)
Alias for `get_report_schedule_list(next_token=token)`.

MWS Docs: GetReportScheduleListByNextToken

Parameters **token** (*str*) –

get_report_schedule_count(*report_types=None*)
Returns a count of order report requests that are scheduled to be submitted to Amazon MWS.

MWS Docs: GetReportScheduleCount

Parameters **report_types** (*List[Union[mws.models.reports.ReportType, str]]*) –

update_report_acknowledgements(*report_ids=None*, *acknowledged=None*)
Updates the acknowledged status of one or more reports.

MWS Docs: UpdateReportAcknowledgements

Parameters

- **report_ids** (*Optional[List[str]]*) –
- **acknowledged** (*Optional[bool]*) –

11.2 Enums

class mws.Reports.**ReportType**(*value*)
Bases: `str`, `enum.Enum`

An enumeration of the types of reports that can be requested from Amazon MWS.

MWS Docs: ReportType enumeration

Available values

You can use either the Enum instance itself or its string value as an argument in relevant request methods. Each of the below examples may be used in a request for a flat file of open listings:

```
from mws import Reports

my_report_type = Reports.ReportType.INVENTORY
# OR
my_report_type = Reports.ReportType.INVENTORY.value
# OR
my_report_type = '_GET_FLAT_FILE_OPEN_LISTINGS_DATA_'
```

INVENTORY = '_GET_FLAT_FILE_OPEN_LISTINGS_DATA_'

ALL_LISTINGS = '_GET_MERCHANT_LISTINGS_ALL_DATA_'

```
ACTIVE_LISTINGS = '_GET_MERCHANT_LISTINGS_DATA_'
INACTIVE_LISTINGS = '_GET_MERCHANT_LISTINGS_INACTIVE_DATA_'
OPEN_LISTINGS = '_GET_MERCHANT_LISTINGS_DATA_BACK_COMPAT_'
OPEN_LISTINGS_LITE = '_GET_MERCHANT_LISTINGS_DATA_LITE_'
OPEN_LISTINGS_LITER = '_GET_MERCHANT_LISTINGS_DATA_LITER_'
CANCELED_LISTINGS = '_GET_MERCHANT_CANCELLED_LISTINGS_DATA_'
SOLD_LISTINGS = '_GET_CONVERGED_FLAT_FILE_SOLD_LISTINGS_DATA_'
LISTING_QUALITY_AND_SUPPRESSED = '_GET_MERCHANT_LISTINGS_DEFECT_DATA_'
PAN_EUROPEAN_ELIGIBILITY_FBA_ASINS = '_GET_PAN_EU_OFFER_STATUS_'
PAN_EUROPEAN_ELIGIBILITY_SELF_FULFILLED_ASINS = '_GET_MFN_PAN_EU_OFFER_STATUS_'
GLOBAL_EXPANSION_OPPORTUNITIES = '_GET_FLAT_FILE_GEO_OPPORTUNITIES_'
REFERRAL_FEE_PREVIEW = '_GET_REFERRAL_FEE_PREVIEW_REPORT_'
ORDERS_UNSHIPPED = '_GET_FLAT_FILE_ACTIONABLE_ORDER_DATA_'
ORDERS_SCHEDULED_XML = '_GET_ORDERS_DATA_'
ORDERS_REQUESTED_OR_SCHEDULED = '_GET_FLAT_FILE_ORDERS_DATA_'
ORDERS_CONVERGED = '_GET_CONVERGED_FLAT_FILE_ORDER_REPORT_DATA_'
TRACKING_BY_LAST_UPDATE = '_GET_FLAT_FILE_ALL_ORDERS_DATA_BY_LAST_UPDATE_'
TRACKING_BY_ORDER_DATE = '_GET_FLAT_FILE_ALL_ORDERS_DATA_BY_ORDER_DATE_'
TRACKING_ARCHIVED_ORDERS_FLATFILE = '_GET_FLAT_FILE_ARCHIVED_ORDERS_DATA_BY_ORDER_DATE
TRACKING_BY_LAST_UPDATE_XML = '_GET_XML_ALL_ORDERS_DATA_BY_LAST_UPDATE_'
TRACKING_BY_ORDER_DATE_XML = '_GET_XML_ALL_ORDERS_DATA_BY_ORDER_DATE_'
PENDING_ORDERS_FLAT_FILE = '_GET_FLAT_FILE_PENDING_ORDERS_DATA_'
PENDING_ORDERS_XML = '_GET_PENDING_ORDERS_DATA_'
PENDING_ORDERS_CONVERGED_FLAT_FILE = '_GET_CONVERGED_FLAT_FILE_PENDING_ORDERS_DATA_'
RETURNS_XML_DATA_BY_RETURN_DATE = '_GET_XML_RETURNS_DATA_BY_RETURN_DATE_'
RETURNS_FLAT_FILE_RETURNS_DATA_BY_RETURN_DATE = '_GET_FLAT_FILE_RETURNS_DATA_BY_RETURN
RETURNS_XML_MFN_PRIME_RETURNS_REPORT = '_GET_XML_MFN_PRIME_RETURNS_REPORT_'
RETURNS_CSV_MFN_PRIME_RETURNS_REPORT = '_GET_CSV_MFN_PRIME_RETURNS_REPORT_'
RETURNS_XML_MFN_SKU_RETURN_ATTRIBUTES_REPORT = '_GET_XML_MFN_SKU_RETURN_ATTRIBUTES_REP
RETURNS_FLAT_FILE_MFN_SKU_RETURN_ATTRIBUTES_REPORT = '_GET_FLAT_FILE_MFN_SKU_RETURN_AT
PERFORMANCE_FEEDBACK = '_GET_SELLER_FEEDBACK_DATA_'
PERFORMANCE_CUSTOMER_METRICS_XML = '_GET_V1_SELLER_PERFORMANCE_REPORT_'
SETTLEMENT_FLATFILE = '_GET_V2_SETTLEMENT_REPORT_DATA_FLAT_FILE_'
SETTLEMENT_V2_XML = '_GET_V2_SETTLEMENT_REPORT_DATA_XML_'
SETTLEMENT_V2_FLATFILE = '_GET_V2_SETTLEMENT_REPORT_DATA_FLAT_FILE_V2_'
FBA_SALES_AMAZON_FULFILLED = '_GET_AMAZON_FULFILLED_SHIPMENTS_DATA_'
```

```
FBA_SALES_ALL_LAST_UPDATE = '_GET_FLAT_FILE_ALL_ORDERS_DATA_BY_LAST_UPDATE_'
FBA_SALES_ALL_BY_ORDER_DATE = '_GET_FLAT_FILE_ALL_ORDERS_DATA_BY_ORDER_DATE_'
FBA_SALES_ALL_BY_LAST_UPDATE_XML = '_GET_XML_ALL_ORDERS_DATA_BY_LAST_UPDATE_'
FBA_SALES_ALL_BY_ORDER_DATE_XML = '_GET_XML_ALL_ORDERS_DATA_BY_ORDER_DATE_'
FBA_SALES_CUSTOMER_SHIPMENT = '_GET_FBA_FULFILLMENT_CUSTOMER_SHIPMENT_SALES_DATA_'
FBA_SALES_PROMOTIONS = '_GET_FBA_FULFILLMENT_CUSTOMER_SHIPMENT_PROMOTION_DATA_'
FBA_SALES_CUSTOMER_TAXES = '_GET_FBA_FULFILLMENT_CUSTOMER_TAXES_DATA_'
FBA_SALES_REMOTE_FULFILLMENT_ELIGIBILITY = '_GET_REMOTE_FULFILLMENT_ELIGIBILITY_'
FBA_INVENTORY_AFN = '_GET_AFN_INVENTORY_DATA_'
FBA_INVENTORY_AFN_BY_COUNTRY = '_GET_AFN_INVENTORY_DATA_BY_COUNTRY_'
FBA_INVENTORY_HISTORY_DAILY = '_GET_FBA_FULFILLMENT_CURRENT_INVENTORY_DATA_'
FBA_INVENTORY_HISTORY_MONTHLY = '_GET_FBA_FULFILLMENT_MONTHLY_INVENTORY_DATA_'
FBA_INVENTORY_RECEIVED = '_GET_FBA_FULFILLMENT_INVENTORY_RECEIPTS_DATA_'
FBA_INVENTORY_RESERVED = '_GET_RESERVED_INVENTORY_DATA_'
FBA_INVENTORY_EVENT_DETAIL = '_GET_FBA_FULFILLMENT_INVENTORY_SUMMARY_DATA_'
FBA_INVENTORY_ADJUSTMENTS = '_GET_FBA_FULFILLMENT_INVENTORY_ADJUSTMENTS_DATA_'
FBA_INVENTORY_HEALTH = '_GET_FBA_FULFILLMENT_INVENTORY_HEALTH_DATA_'
FBA_INVENTORY_MANAGE_ACTIVE = '_GET_FBA_MYI_UNSUPPRESSED_INVENTORY_DATA_'
FBA_INVENTORY_MANAGE_ALL = '_GET_FBA_MYI_ALL_INVENTORY_DATA_'
FBA_INVENTORY_RESTOCK_INVENTORY = '_GET_RESTOCK_INVENTORY_RECOMMENDATIONS_REPORT_'
FBA_INVENTORY_CROSS_BORDER_MOVEMENT = '_GET_FBA_FULFILLMENT_CROSS_BORDER_INVENTORY_MOV
FBA_INVENTORY_INBOUND_PERFORMANCE = '_GET_FBA_FULFILLMENT_INBOUND_NONCOMPLIANCE_DATA_'
FBA_INVENTORY_STRANDED = '_GET_STRANDED_INVENTORY_UI_DATA_'
FBA_INVENTORY_BULK_FIX_STRANDED = '_GET_STRANDED_INVENTORY_LOADER_DATA_'
FBA_INVENTORY_AGE = '_GET_FBA_INVENTORY_AGED_DATA_'
FBA_INVENTORY_EXCESS = '_GET_EXCESS_INVENTORY_DATA_'
FBA_INVENTORY_STORAGE_FEE_CHARGES = '_GET_FBA_STORAGE_FEE_CHARGES_DATA_'
FBA_INVENTORY_PRODUCT_EXCHANGE = '_GET_PRODUCT_EXCHANGE_DATA_'
FBA_PAYMENTS_FEE_PREVIEW = '_GET_FBA_ESTIMATED_FBA_FEES_TXT_DATA_'
FBA_PAYMENTS_REIMBURSEMENTS = '_GET_FBA_REIMBURSEMENTS_DATA_'
FBA_PAYMENTS_LONGTERM_STORAGE_FEE_CHARGES = '_GET_FBA_FULFILLMENT_LONGTERM_STORAGE_FEE_
FBA_CONCESSION_RETURNS = '_GET_FBA_FULFILLMENT_CUSTOMER_RETURNS_DATA_'
FBA_CONCESSION_SHIPMENT_REPLACEMENT = '_GET_FBA_FULFILLMENT_CUSTOMER_SHIPMENT_REPLACEM
FBA_REMOVAL_RECOMMENDED = '_GET_FBA_RECOMMENDED_REMOVAL_DATA_'
FBA_REMOVAL_ORDER_DETAIL = '_GET_FBA_FULFILLMENT_REMOVAL_ORDER_DETAIL_DATA_'
FBA_REMOVAL_SHIPMENT_DETAIL = '_GET_FBA_FULFILLMENT_REMOVAL_SHIPMENT_DETAIL_DATA_'
```

```
FBA_SMALL_LIGHT_INVENTORY = '_GET_FBA_UNO_INVENTORY_DATA_'
SALES_TAX = '_GET_FLAT_FILE_SALES_TAX_DATA_'
VAT_CALCULATION = '_SC_VAT_TAX_REPORT_'
VAT_TRANSACTIONS = '_GET_VAT_TRANSACTION_DATA_'
TAX_GST_MERCHANT_B2B = '_GET_GST_MTR_B2B_CUSTOM_'
TAX_GST_MERCHANT_B2C = '_GET_GST_MTR_B2C_CUSTOM_'
BROWSE_TREE = '_GET_XML_BROWSE_TREE_DATA_'
EASYSHIP_DOCUMENTS = '_GET_EASYSHIP_DOCUMENTS_'
EASYSHIP_PICKED_UP = '_GET_EASYSHIP_PICKEDUP_'
EASYSHIP_WAITING_FOR_PICKUP = '_GET_EASYSHIP_WAITING_FOR_PICKUP_'
AMZN_BUSINESS_FEE_DISCOUNTS_REPORT = '_FEE_DISCOUNTS_REPORT_'
AMZN_BUSINESS_RFQD_BULK_DOWNLOAD = '_RFQD_BULK_DOWNLOAD_'
AMAZONPAY_SANDBOX_SETTLEMENT = '_GET_FLAT_FILE_OFFAMAZONPAYMENTS_SANDBOX_SETTLEMENT_DA
```

class mws.Reports.**Schedule**(*value*)

Bases: str, enum.Enum

An enumeration of the units of time that reports can be requested.

MWS Docs: Schedule enumeration

Available values

Several schedule frequencies are provided by Amazon, and this Enum provides easy access to all of them through several aliases for each schedule type.

```
EVERY_15_MIN = '_15_MINUTES_'
EVERY_15_MINS = '_15_MINUTES_'
EVERY_15_MINUTE = '_15_MINUTES_'
EVERY_15_MINUTES = '_15_MINUTES_'
EVERY_30_MIN = '_30_MINUTES_'
EVERY_30_MINS = '_30_MINUTES_'
EVERY_30_MINUTE = '_30_MINUTES_'
EVERY_30_MINUTES = '_30_MINUTES_'
EVERY_HOUR = '_1_HOUR_'
EVERY_1_HOUR = '_1_HOUR_'
EVERY_1_HOURS = '_1_HOUR_'
EVERY_2_HOUR = '_2_HOURS_'
EVERY_2_HOURS = '_2_HOURS_'
EVERY_4_HOUR = '_4_HOURS_'
EVERY_4_HOURS = '_4_HOURS_'
EVERY_8_HOUR = '_8_HOURS_'
```

EVERY_8_HOURS = '_8_HOURS_'

EVERY_12_HOUR = '_12_HOURS_'

EVERY_12_HOURS = '_12_HOURS_'

DAILY = '_1_DAY_'

EVERY_DAY = '_1_DAY_'

EVERY_1_DAY = '_1_DAY_'

EVERY_1_DAYS = '_1_DAY_'

EVERY_2_DAY = '_2_DAYS_'

EVERY_2_DAYS = '_2_DAYS_'

EVERY_48_HOUR = '_2_DAYS_'

EVERY_48_HOURS = '_2_DAYS_'

EVERY_3_DAY = '_72_HOURS_'

EVERY_3_DAYS = '_72_HOURS_'

EVERY_72_HOUR = '_72_HOURS_'

EVERY_72_HOURS = '_72_HOURS_'

WEEKLY = '_1_WEEK_'

EVERY_WEEK = '_1_WEEK_'

EVERY_1_WEEK = '_1_WEEK_'

EVERY_1_WEEKS = '_1_WEEK_'

EVERY_7_DAY = '_1_WEEK_'

EVERY_7_DAYS = '_1_WEEK_'

EVERY_14_DAY = '_14_DAYS_'

EVERY_14_DAYS = '_14_DAYS_'

EVERY_2_WEEK = '_14_DAYS_'

EVERY_2_WEEKS = '_14_DAYS_'

FORTNIGHTLY = '_14_DAYS_'

EVERY_15_DAY = '_15_DAYS_'

EVERY_15_DAYS = '_15_DAYS_'

EVERY_30_DAY = '_30_DAYS_'

EVERY_30_DAYS = '_30_DAYS_'

DELETE = '_NEVER_'

Delete a previously created report request schedule.

class mws.Reports.**ProcessingStatus**(*value*)

Bases: str, enum.Enum

An optional enumeration of common processing_status values.

SUBMITTED = '_SUBMITTED_'

IN_PROGRESS = '_IN_PROGRESS_'

CANCELLED = '_CANCELLED_'

CANCELED = '_CANCELLED_'

An alias for “CANCELLED”, as some folks spell it with one L and there’s nothing wrong with that.

DONE = '_DONE_'

DONE_NO_DATA = '_DONE_NO_DATA_'

CHAPTER

TWELVE

DOTDICT

New in version 1.0dev15: `DotDict` added.

Warning: The following pertains to features added in **v1.0dev15** related to MWS requests. These features are disabled by default. To use these features, set flag `_use_feature_mwsresponse` to `True` on an API class instance *before* making any requests:

```
api_class = Orders(...)
api_class._use_feature_mwsresponse = True
```

If the flag is `False`, all requests will return either `DictWrapper` or `DataWrapper` objects (deprecated); and parsed XML contents will be returned as an instance of `ObjectDict` (deprecated).

New features using `MWSResponse` *and* `DotDict` *will become the default in v1.0.*

The *`DotDict`* class is a subclass of a standard Python dict that provides access to its keys as attributes. This object is used mainly for parsed XML content returned by *`MWSResponse.parsed`* and *`MWSResponse.metadata`*, but `DotDict` can also be used as a general-purpose `dict` replacement (with some caveats, as shown below).

12.1 Keys as attributes

While keys of a `DotDict` can be accessed the same as keys in a standard `dict`, they can also be accessed as attributes:

```
from mws.utils.collections import DotDict

foo = DotDict({'spam': 'ham'})

print(foo['spam'])
# 'ham'
print(foo.spam)
# 'ham'
print(foo.get('spam'))
# 'ham'
```

This is useful for traversing the nested structures created by parsing XML documents, where several keys are required in order to access a leaf node.

Consider the following (truncated and edited) example response from the MWS operation `ListMatchingProducts`:

```
<?xml version="1.0"?>
<ListMatchingProductsResponse xmlns="http://mws.amazonservices.com/schema/Products/
↪2011-10-01">
  <ListMatchingProductsResult>
    <Products>
      <Product>
        <Identifiers>
          <MarketplaceASIN>
            <MarketplaceId>ACBDEFGH</MarketplaceId>
            <ASIN>B0987654</ASIN>
          </MarketplaceASIN>
        </Identifiers>
      </Product>
    </Products>
  </ListMatchingProductsResult>
</ListMatchingProductsResponse>
```

When this document is parsed, accessing the `<ASIN>` on line 9 using dict keys looks like the following:

```
# assuming `response` is an instance of `MWSResponse`
asin = response.parsed['Products']['Product']['Identifiers']['MarketplaceASIN']['ASIN
↪']
```

Using attribute access, the above call turns into:

```
asin = response.parsed.Products.Product.Identifiers.MarketplaceASIN.ASIN
```

And, of course, a mix of the different methods is possible:

```
asin = response.parsed['Products'].get('Product').Identifiers['MarketplaceASIN'].get(
↪'ASIN')
```

Tip: Accessing specific data in an MWS response will often produce lengthy code lines, as the above samples show. We recommend following best practices for Python programs in general, breaking up these longer lines by assigning chunks of data to intermediary variables:

```
product = response.parsed.Products.Product
asin = product.Identifiers.MarketplaceASIN.ASIN
```

12.2 Native iteration

XML represents sequences of similar objects by having sibling tags with the same tag name. Consider the following toy example with three `<Product>` tags:

```
<Response>
  <Products>
    <Product>
      <Name>spam</Name>
    </Product>
    <Product>
      <Name>ham</Name>
    </Product>
```

(continues on next page)

(continued from previous page)

```
    <Product>
      <Name>eggs</Name>
    </Product>
  </Products>
</Response>
```

When parsed, these are collected into a list of `DotDict` instances:

```
DotDict({
    'Products': DotDict({
        'Product': [
            DotDict({'Name': 'spam'}),
            DotDict({'Name': 'ham'}),
            DotDict({'Name': 'eggs'}),
        ]
    })
})
```

Note: The list of objects will always be found under the same key name as the duplicate tags, i.e. `Product`; *not* under their parent key, `Products`. This may seem counterintuitive, but the parser is simply preserving all tag names present in the XML document.

Further, if a tag attribute is present on the parent `<Products>` tag, you would be able to access it as a separate key at the same level as `Product`. This would not be possible if `Products` returned a list.

To gather the names of all products in this response, we can simply iterate over this list:

```
names = []
for product in response.parsed.Products.Product:
    names.append(product.Name)

print(names)
# ['spam', 'ham', 'eggs']
```

If the same request returns only one `<Product>` tag, the `Product` key in the parsed response will return only a single `DotDict`, similar to any other node in the XML tree. Trying to access the `Product` node in this case as though it were a list - such as using indices (`.Product[0]`) - will result in errors.

However, when a `DotDict` is iterated, it will wrap itself in a list in order to provide the same interface as before.

So, for an XML response like so:

```
<Response>
  <Products>
    <Product>
      <Name>foo</Name>
    </Product>
  </Products>
</Response>
```

. . . the same Python code can be used to access "all" `Product` keys:

```
names = []
for product in response.parsed.Products.Product:
    names.append(product.Name)
```

(continues on next page)

(continued from previous page)

```
print(names)
# ['foo']
```

Note: While `DotDict` is a subclass of `dict`, this behavior is different from that of the standard `dict`, where iterating directly on the `dict` object is equivalent to iterating on `dict.keys()`. We have chosen to implement the above behavior to more closely match most users' intended usage when working with parsed XML, even though `DotDict` *can* be used much like a standard `dict` for (most) general purposes.

12.3 Recursive conversion of dict objects

`DotDict` instances expect to hold nested data, as seen in the examples throughout this document. As such, any `dict` assigned as a value to a `DotDict` is automatically converted to a `DotDict`, as well. The values of the assigned `dict` are then recursively built the same way, such that every `dict` (or other mapping type) instance in the structure is also converted to `DotDict`.

This holds true in a variety of scenarios:

- Wrapping a nested `dict` in `DotDict`:

```
example1 = DotDict({'spam': {'ham': {'eggs': 'juice'}}})
print(example1)
# DotDict({'spam': DotDict({'ham': DotDict({'eggs': 'juice'})})})
```

- Using kwargs to build `DotDict`, with a `dict` as one of the values:

```
example2 = DotDict(spam={'muffin': {'cereal': 'milk'}})
print(example2)
DotDict({'spam': DotDict({'muffin': DotDict({'cereal': 'milk'})})})
```

- Assigning a `dict` to a key of an existing `DotDict`, including creating new keys:

```
example3 = DotDict()
example3.pancakes = {'maple': 'syrup'}
print(example3)
# DotDict({'pancakes': DotDict({'maple': 'syrup'})})

example3.pancakes.toast = {'strawberry': 'jam'}
print(example3)
# DotDict({'pancakes': DotDict({'maple': 'syrup', 'toast': DotDict({'strawberry':
↪'jam'})})})
```

- Using `DotDict.update` in a similar manner as `dict.update`:

```
example4 = DotDict()
example4.update({'chicken': {'waffles': 'honey'}})
print(example4)
# DotDict({'chicken': DotDict({'waffles': 'honey'})})

# Including a mix of a plain dict and kwargs
example5 = DotDict()
example5.update({'running': {'out': 'of'}}, food='examples', to={'use': 'here'})
```

(continues on next page)

(continued from previous page)

```
print(example5)
# DotDict({'running': DotDict({'out': 'of'}), 'food': 'examples', 'to': DotDict({
↪'use': 'here'})})
```

12.4 Working with XML tag attributes

`DotDict` is used in python-amazon-mws primarily for parsed XML content. As such, some features of the class are specialized for working with that content.

XML tags can contain attributes with additional data points. When parsed, these attributes are assigned to their own dict keys starting with `@`, differentiating them from normal tag names.

Further, tags that contain an attribute and text content will store the text on a special key, `#text`.

For example, with the following XML document:

```
<Response>
  <Products>
    <Product Name="spam">
      <SomethingElse>ham</SomethingElse>
      <WhatHaveYou anotherAttr="foo">eggs</WhatHaveYou>
    </Product>
  </Products>
</Response>
```

The parsed response would look like:

```
DotDict({
    'Products': DotDict({
        'Product': DotDict({
            '@Name': 'spam',
            'SomethingElse': 'ham',
            'WhatHaveYou': DotDict({
                '@anotherAttr': 'foo',
                '#text': 'eggs'
            })
        })
    })
})
```

These `@` and `#text` keys cannot be accessed directly as attributes due to Python syntax, which reserves the `@` and `#` characters. You can still use standard dict keys to access this content:

```
print(dotdict.Products.Product['@Name'])
# 'spam'

print(dotdict.Products.Product.WhatHaveYou['#text'])
# 'eggs'
```

`DotDict` also allows accessing these keys using a fallback method. Simply provide the key name without `@` or `#` in front, and it will attempt to find a matching key:

```
print(dotdict.Products.Product.Name)
# 'spam'
```

(continues on next page)

(continued from previous page)

```
print(dotdict.Products.Product.WhatHaveYou.text)
# 'eggs'
```

Note: In case of a conflicting key name, a key matching the attribute will be returned first:

```
dotdict = DotDict({'foo': 'spam', '@foo': 'ham'})
print(dotdict.foo)
# 'spam'
print(dotdict['@foo'])
# 'ham'
```

This conflict is a rare occurrence for most XML documents, however, as they are not likely to return a tag attribute with the same name as an immediate child tag.

12.5 DotDict API

class mws.**DotDict**(**args*, ***kwargs*)

Read-only dict-like object class that wraps a mapping object.

New in version 1.0dev15.

update(**args*, ***kwargs*)

Recursively builds values in any nested objects, such that any mapping object in the nested structure is converted to a `DotDict`.

- Each nested mapping object will be converted to `DotDict`.
- Each non-string, non-dict iterable will have elements built, as well.
- All other objects in the data are left unchanged.

classmethod build(*obj*)

Builds objects to work as recursive versions of this object.

- Mappings are converted to a DotDict object.
- For iterables, each element in the sequence is run through the build method recursively.
- All other objects are returned unchanged.

CHAPTER

THIRTEEN

MWSRESPONSE

New in version 1.0dev15: `MWSResponse` added

Warning: The following pertains to features added in **v1.0dev15** related to MWS requests. These features are disabled by default. To use these features, set flag `_use_feature_mwsresponse` to `True` on an API class instance *before* making any requests:

```
api_class = Orders(...)
api_class._use_feature_mwsresponse = True
```

If the flag is `False`, all requests will return either `DictWrapper` or `DataWrapper` objects (deprecated); and parsed XML contents will be returned as an instance of `ObjectDict` (deprecated).

New features using `MWSResponse` *and* `DotDict` *will become the default in v1.0.*

`MWSResponse` acts as a wrapper for `requests.Response` objects returned from requests made to MWS. When initialized, the response content is *automatically parsed for XML content*, making it available as a `DotDict` instance in *`MWSResponse.parsed`*.

13.1 Parsed content for XML responses

All XML response content is automatically parsed using the `xmltodict` package. The parsed results are stored as a *`DotDict`* accessible from *`MWSResponse.parsed`*.

For more details on working with the parsed content, please see *DotDict*.

13.2 Original response access

As `MWSResponse` wraps a `requests.Response` object, all data and methods of that underlying object can be accessed from the `MWSResponse` instance using one of the following:

- The *`MWSResponse.original`* attribute:

```
response = api.foo_request(...)
# response is an instance of MWSResponse

response.original.status_code
# 200
response.original.headers
# {'Content-Type': ...}
```

(continues on next page)

(continued from previous page)

```
response.original.text  # unicode
# 'Hello world!'
response.original.content  # bytes
# b'Hello world!'
```

- A number of shortcut properties available on `MWSResponse` itself:

```
response.content      # response.original.content
response.cookies      # response.original.cookies
response.elapsed      # response.original.elapsed
response.encoding     # response.original.encoding
response.headers      # response.original.headers
response.reason       # response.original.reason
response.request      # response.original.request
response.status_code  # response.original.status_code
response.text         # response.original.text
```

Each of these shortcuts is a read-only property, with the exception of `response.encoding`, which includes a setter for convenience when dealing with content encoding issues:

```
response.encoding = "iso-8859-1"
print(response.original.encoding)
# "iso-8859-1"
```

13.3 MWSResponse API

New in version 1.0dev15.

class mws.**MWSResponse**(*response*, *result_key=None*, *encoding=None*, *force_cdata=False*)

Wraps a `requests.Response` object and extracts some known data.

Particularly for XML responses, parsed contents can be found in the `.parsed` property as a `DotDict` instance.

Find metadata in `.metadata`, mainly for accessing `.metadata.RequestId`; or simply use the `.request_id` shortcut attr.

Parameters

- **response** (*request.Response*) – Response object returned by a request sent to MWS.
- **result_key** (*str*) – Key to use as the root for `.parsed`. Typically a tag in the root of the response's XML document whose name ends in `Result`. Defaults to `None`, in which case the full document is presented when using `.parsed`.
- **force_cdata** (*bool*) – Passed to `xmltodict.parse()` when parsing the response's XML document. Defaults to `False`.

original: requests.Response

Instance of the original `requests.Response` object. Can be used to get or set data in the original response.

property encoding

Shortcut to `.original.encoding`. Can also be used as a setter, changing the encoding of the response. This then changes how content is decoded when using *.text*.

parse_response(*force_cdata=False*)
Runs *.text* through `xmltodict.parse()`, storing the returned Python dictionary as `._dict`.

If no XML errors occur during that process, constructs `DotDict` instances from the parsed XML data, making them available from *.parsed* and *.metadata*.

For non-XML responses, does nothing.

Parameters **force_cdata** (*bool*) – Passed to `xml_to_dict.parse()` when parsing XML content. Defaults to `False`. Ignored for non-XML responses.

property parsed
Returns a parsed version of the response.

For XML documents, returns a `DotDict` of the parsed XML content, starting from `._result_key`.

For all other types of responses, returns *.text* instead.

property metadata
Returns a `DotDict` instance from the response's `ResponseMetadata` key, if present. Typically the only key of note here is `.metadata.RequestId`, which can also be accessed with *.request_id*.

property content
Shortcut to `.original.content`, which is bytes.

property cookies
Shortcut to `.original.cookies`.

property elapsed
Shortcut to `.original.elapsed`.

property headers
Shortcut to `.original.headers`.

property reason
Shortcut to `.original.reason`.

property request
Shortcut to `.original.request`.

property request_id
Returns the value of a `RequestId` from *.metadata*, if present, otherwise `None`.

property status_code
Shortcut to `.original.status_code`.

property text
Shortcut to `.original.text`, which is unicode.

PYTHON MODULE INDEX

m

INDEX

F

G

H

I

J

L

M

N

O

P

W