python-amazon-product-api Documentation

Release 0.2.8

Sebastian Rahlf

September 05, 2015

Contents

The Amazon Product Advertising API provides programmatic access to Amazon's product selection and discovery functionality. It has search and look up capabilities, provides information on products and other features such as Reviews, Similar Products and New and Used listings. `python-amazon-product-api` offers a light-weight access to the latest version of the API without getting in your way.

CHAPTER 1

Installation

The easiest way to get the Python bindings is using pip.

```
pip install python-amazon-product-api
```

Alternatively ou can download the .tgz file from Cheeseshop, untar it and run:

```
python setup.py install
```

You'll also find binaries there to make your life easier if you happen to use a Windows system.

The development version is available on bitbucket.org. Feel free to clone the repository and add your own features (see also *How to contribute*).

```
hg clone http://bitbucket.org/basti/python-amazon-product-api/
```

If you like what you see, drop me a line at *basti at redtoad dot de*.

CHAPTER 2

Getting started

In order to use this API you'll obviously need an Amazon Associates Web Service account for which you must register with Amazon. Each account contains an *AWSAccessKeyId* and a *SecretKey*. As of API version 2011-08-01 you will also need to register for an AssociateTag.

Note: It is assumed that you know what the Amazon Product Advertising API does. If you are unsure, read their developer guide (particularly the section *Introduction to the Product Advertising API*.

2.1 Basic setup

If you haven't done so already, create a file `~/.amazon-product-api` (`C:\Users\You\.amazon-product-api` if you're on Windows) and paste the following content into it:

```
[Credentials]
access_key = <your access key>
secret_key = <your secret key>
associate_tag = <your associate id>
```

Of course, you'll need to fill in the appropriate values! More information on how to configure the module can be found *later on*.

2.2 Your first API request

Here is an example how to use the API to search for books of a certain publisher.

```
api = API(locale='us')
items = api.item_search('Books', Publisher="O'Reilly")
```

So what happens here? First you initialised your API wrapper to use Amazon.com. There are, of course, other locales available should you wish to use a different one. For instance, `locale='de'` will cause requests to be sent to Amazon.de (Germany).

Afterwards you called the API operation *ItemSearch* to get a list of all books that where published by O'Reilly. Now method `item_search` does several things at once for you:

1. It turns all your parameters into a validly signed URL and sends a request.
2. The returned XML document is parsed and if it contains any error message, the appropriate Python exception is raised (see *Dealing with errors*).

3. Amazon itself provides their results spread over several pages. If you were to do this manually you would have to make several calls. To make things easier for you `item_search()` will iterate over all availabe results (see *Result pagination* for more information).

You can now iterate over the `items` and will get a number of parsed XML nodes (by default and if available lxml.objectify is used). With it you can access all elements and attributes in a Pythonic way:

```
# get all books from result set and
# print author and title
for book in items:
    print '%s: "%s"' % (book.ItemAttributes.Author,
                        book.ItemAttributes.Title)
```

Please refer to the lxml.objectify documentation for more details. If you cannot/will not use lxml, see *Result processing* for alternatives.

You can find more API operations later in *Operations*.

2.3 Dealing with errors

One of the advatages of using this wrapper is that all error messages from Amazon will raise Python exceptions with meaningful messages.

```
try:
    node = api.similarity_lookup('0451462009', '0718155157')
    # ...
except NoSimilarityForASIN, e:
    print 'There is no book similar to %s!' % e.args[0]
except AWSError, e:
    print 'Amazon complained about yout request!'
    print e.code
    print e.msg
```

A list of exceptions can be found in *Error handling*.

CHAPTER 3

Operations

All functionality of the Amazon Product Advertising API is provided by *operations* each of which will accept a number of different parameters both required and optional. A special signed URL has to be constructed from which the result of an operation can be retrieved as a XML document.

Building the individual URL can be quite cumbersome when done repeatedly by hand. That's the main reason why this module came into being. Any operation listed in the API documentation can thus be called with `call()`. To look up information on an article, one could for instance call ItemLookup in the following way:

```
api.call(Operation='ItemLookup', ItemId='B00008OE6I')
```

However, this module offers a few *convenience methods* which can make your life easier by producing clearer error messages or even *paginating* over the returned results. For the above call you would simply use *`item_lookup()`*.

Below is a list of all the operations which are specifically supported in this module.

3.1 Lookup and search operations

These operations are the heart and soul of the API. With these you can search for products and retreive their data.

`API.`**`item_search`**(*searchindex*, ***query*)

Changed in version 2011-08-01: You can only fetch up to 10 result pages (instead of 400).

The `item_search()` operation returns items that satisfy the search criteria, including one or more search indices.

`item_search()` returns up to ten search results at a time. When `condition` equals "All," `item_search()` returns up to three offers per condition (if they exist), for example, three new, three used, three refurbished, and three collectible items. Or, for example, if there are no collectible or refurbished offers, `item_search()` returns three new and three used offers.

Because there are thousands of items in each search index, `item_search()` requires that you specify the value for at least one parameter in addition to a search index. The additional parameter value must reference items within the specified search index. For example, you might specify a browse node (`BrowseNode` is an `item_search()` parameter), Harry Potter Books, within the Books product category. You would not get results, for example, if you specified the search index to be Automotive and the browse node to be Harry Potter Books. In this case, the parameter value is not associated with the search index value.

The `ItemPage` parameter enables you to return a specified page of results. The maximum `ItemPage` number that can be returned is 400. An error is returned if you try to access higher numbered pages. If you do not include `ItemPage` in your request, the first page will be returned by default. There can be up to ten items per page (see *Result pagination* for more details).

`item_search()` is the operation that is used most often in requests. In general, when trying to find an item for sale, you use this operation.

Examples:

- Use the search index, Toys, and the parameter, Keywords, to return information about all toy rockets sold in by Amazon.

```
>>> api.item_search('Toys', Keywords='Rocket')
```

- Use a blended search to look through multiple search indices for items that have "Mustang" in their name or description. A blended search looks through multiple search indices at the same time.

```
>>> api.item_search('Blended', Keywords='Mustang')
```

- Use the Availability parameter to only return shirts that are available:

```
>>> api.item_search('Apparel', Condition='All',
...     Availability='Available', Keywords='Shirt')
```

- Set the search index to `MusicTracks` and `Keywords` to the title of a song to find a song title.
- Use the `BrowseNodes` response group to find the browse node of an item.
- Use the `Variations` response group and the `BrowseNode` parameter to find all of the variations of a parent browse node.

`API.`**`item_lookup`**(*id*[, *id2*, ...], ***extra*)

Given an item identifier, the `item_lookup()` operation returns some or all of the item attributes, depending on the response group specified in the request. By default, `item_lookup()` returns an item's ASIN, Manufacturer, ProductGroup, and Title of the item.

```
>>> api = API(locale='uk')
>>> result = api.item_lookup('B006H3MIV8')
>>> for item in result.Items.Item:
...     print '%s (%s) in group %s' % (
...         item.ItemAttributes.Title, item.ASIN)
...
Chimes of Freedom: The Songs of Bob Dylan (B006H3MIV8)
```

`item_lookup()` supports many response groups, so you can retrieve many different kinds of product information, called item attributes, including product reviews, variations, similar products, pricing, availability, images of products, accessories, and other information.

To look up more than one item at a time, you can pass several identifiers at once.

```
>>> res = api.item_lookup('B000002O4S', 'B000002O6R', 'B0000014RN')
```

Note: The parameter support varies by locale used.

Examples:

- The following request returns the information associated with ItemId B00008OE6I.

```
>>> api.item_lookup('B00008OE6I')
```

- The following request returns an offer for a refurbished item that is not sold by Amazon

```
>>> api.item_lookup('B00008OE6I',
...     ResponseGroup='OfferFull', Condition='All')
```

- In the following request, the ItemId is an SKU, which requires that you also specify the IdType.

```
>>> api.item_lookup([SKU], IdType='SKU')
```

- If you use a UPC as ItemId, you also need to specify SearchIndex and ItemType.

```
>>> api.item_lookup([UPC], SearchIndex='Books', IdType='UPC')
```

In the following request, the ItemId is an EAN, which requires that you also specify the SearchIndex and ItemType.

```
>>> api.item_lookup([EAN], IdType='EAN')
```

Tips:

- Use the `BrowseNodes` response group to find the browse node of an item.
- Use the `Tracks` response group to find the track, title, and number for each track on each CD in the response.
- Use the `Similarities` response group to find the ASIN and Title for similar products returned in the response.
- Use the `Reviews` response group to find reviews written by customers about an item, and the total number of reviews for each item in the response.
- Use the `OfferSummary` response group to find the number of offer listings and the lowest price for each of the offer listing condition classes, including `New`, `Used`, `Collectible`, and `Refurbished`.
- Use the `Accessories` response group to find the a list of accessory product ASINs and Titles for each product in the response that has accessories.
- The following requests an iframe that contains customer reviews for the specified item.

```
>>> api.item_lookup('0316067938', ResponseGroup='Reviews',
...     TruncateReviewsAt=256, IncludeReviewsSummary=False)
```

`API.`**`similarity_lookup`**(*id*[, *id2*, ...], ***extra*)

The `similarity_lookup()` operation returns up to ten products per page that are similar to one or more items specified in the request. This operation is typically used to pique a customer's interest in buying something similar to what they've already ordered.

If you specify more than one item, `similarity_lookup()` returns the intersection of similar items each item would return separately. Alternatively, you can use the SimilarityType parameter to return the union of items that are similar to any of the specified items. A maximum of ten similar items are returned; the operation does not return additional pages of similar items. If there are more than ten similar items, running the same request can result in different answers because the ten that are included in the response are picked randomly. The results are picked randomly only when you specify multiple items and the results include more than ten similar items.

When you specify multiple items, it is possible for there to be no intersection of similar items. In this case, the operation raises the exception *`NoSimilarityForASIN`*.

This result is very often the case if the items belong to different search indices. The error can occur, however, even when the items share the same search index.

Similarity is a measurement of similar items purchased, that is, customers who bought X also bought Y and Z. It is not a measure, for example, of items viewed, that is, customers who viewed X also viewed Y and Z.

Items returned can be filtered by:

Condition Describes the status of an item. Valid values are `All`, `New` (default), `Used`, `Refurbished` or `Collectible`. When the Availability parameter is set to "Available," the Condition parameter cannot be set to "New."

Examples:

•Return items that are similar to a list of items.

```
>>> api.similarity_lookup('ASIN1', 'ASIN2', 'ASIN3')
```

This request returns the intersection of the similarities for each ASIN. The response to this request is shown in Response to Sample Request.

Return up to ten items that are similar to any of the ASINs specified.

```
>>> api.similarity_lookup('ASIN1', 'ASIN2', 'ASIN3',
...     SimilarityType='Random')
```

This request returns the union of items that are similar to all of the ASINs specified. Only ten items can be returned and those are picked randomly from all of the similar items. Repeating the operation could produce different results.

Parameters **ids** – One or more ASINs you want to look up. You can specify up to ten Ids.

Amazon als structures their products in categories, so called *BrowseNodes*, each with its unique ID. You can find a list of these nodes here.

`API.`**`browse_node_lookup`**(*browse_node_id*, *response_group=None*, ***params*)

Given a `browse_node_id`, this method returns the specified browse node's name, children, and ancestors. The names and browse node IDs of the children and ancestor browse nodes are also returned. `browse_node_lookup()` enables you to traverse the browse node hierarchy to find a browse node.

As you traverse down the hierarchy, you refine your search and limit the number of items returned. For example, you might traverse the following hierarchy: `Books>Children's Books>Science`, to select out of all the science books offered by Amazon only those that are appropriate for children:

```
>>> api = API(locale='us')
>>> node_id = 3207 # Books > Children's Books > Science
>>> result = api.browse_node_lookup(node_id)
>>> for child in result.BrowseNodes.BrowseNode.Children.BrowseNode:
...     print '%s (%sa)' % (child.Name, child.BrowseNodeId)
...
Agriculture (3208)
Anatomy & Physiology (3209)
Astronomy & Space (3210)
Biology (3214)
Botany (3215)
Chemistry (3216)
Earth Sciences (3217)
Electricity & Electronics (3220)
Engineering (16244041)
Environment & Ecology (3221)
Experiments & Projects (3224)
Geography (16244051)
Health (3230)
Heavy Machinery (3249)
How Things Work (3250)
Inventions & Inventors (16244711)
Light & Sound (16244701)
Math (3253)
```

```
Mystery & Wonders (15356851)
Nature (3261)
Physics (3283)
Social Science (3143)
Zoology (3301)
```

Returning the items associated with children's science books produces a much more targeted result than a search based at the level of books.

Alternatively, by traversing up the browse node tree, you can determine the root category of an item. You might do that, for example, to return the top seller of the root product category using the `TopSellers` response group in an `browse_node_lookup()` request:

```
>>> # extract all category roots
>>> result = api.item_lookup('031603438X', # Keith Richards: Life
...     ResponseGroup='BrowseNodes')
>>> root_ids = result.xpath(
...     '//aws:BrowseNode[aws:IsCategoryRoot=1]/aws:BrowseNodeId',
...     namespaces={'aws': result.nsmap.get(None)})

>>> # TopSellers for first category
>>> result = api.browse_node_lookup(root_ids[0], 'TopSellers')
>>> for item in result.BrowseNodes.BrowseNode.TopSellers.TopSeller:
...     print item.ASIN, item.Title
...
B004LLHE62 Ghost in the Polka Dot Bikini (A Ghost of Granny Apples Mystery)
B004LROUNG The Litigators
B005K0HDGE 11/22/63 [Enhanced eBook]
B004W2UBYW Steve Jobs
1419702238 Diary of a Wimpy Kid: Cabin Fever
1451648537 Steve Jobs
B003YL4LNY Inheritance (The Inheritance Cycle)
0375856110 Inheritance (The Inheritance Cycle)
B005IGVS6Q Unfinished Business
B005O548QI Last Breath
```

You can use `browse_node_lookup()` iteratively to navigate through the browse node hierarchy to reach the node that most appropriately suits your search. Then you can use the browse node ID in an `item_search()` request. This response would be far more targeted than, for example, searching through all of the browse nodes in a search index.

A list of BrowseNodes can be found here: http://docs.amazonwebservices.com/AWSECommerceService/latest/DG/index.html?Br

Parameters

- **`browse_node_id`** (*positive int*) – A positive integer assigned by Amazon that uniquely identifies a product category.
- **`response_group`** (*str*) – Specifies the types of values to return. You can specify multiple response groups in one request by separating them with commas. Valid Values are `BrowseNodeInfo` (default), `MostGifted`, `NewReleases`, `MostWishedFor`, `TopSellers`.
- **`params`** – This can be any (or none) of the *Common request parameters*.

3.2 Cart operations

Since the Amazon Product Advertising API is all about generating revenue for Amazon, of course, there is also the possibility to create remote shopping baskets. The operations below are staight-forward and need little explanation. You may, however, have a look at the `amazonproduct.contrib.cart` module which provides a generic `Cart` class to deal with the responses from these operations.

`API.`**`cart_create`**(*items*, ***params*)

`cart_create()` enables you to create a remote shopping cart. A shopping cart is the metaphor used by most e-commerce solutions. It is a temporary data storage structure that resides on Amazon servers. The structure contains the items a customer wants to buy. In Product Advertising API, the shopping cart is considered remote because it is hosted by Amazon servers. In this way, the cart is remote to the vendor's web site where the customer views and selects the items they want to purchase.

Once you add an item to a cart by specifying the item's ListItemId and ASIN, or `OfferListingId`, the item is assigned a `CartItemId` and accessible only by that value. That is, in subsequent requests, an item in a cart cannot be accessed by its `ListItemId` and `ASIN`, or `OfferListingId`. `CartItemId` is returned by `cart_create()`, `cart_get()`, and `cart_add()`.

Because the contents of a cart can change for different reasons, such as item availability, you should not keep a copy of a cart locally. Instead, use the other cart operations to modify the cart contents. For example, to retrieve contents of the cart, which are represented by `CartItemIds`, use `cart_get()`.

Available products are added as cart items. Unavailable items, for example, items out of stock, discontinued, or future releases, are added as `SaveForLaterItems`. No error is generated. The Amazon database changes regularly. You may find a product with an offer listing ID but by the time the item is added to the cart the product is no longer available. The checkout page in the Order Pipeline clearly lists items that are available and those that are `SaveForLaterItems`.

It is impossible to create an empty shopping cart. You have to add at least one item to a shopping cart using a single `cart_create()` request. You can add specific quantities (up to 999) of each item.

`cart_create()` can be used only once in the life cycle of a cart. To modify the contents of the cart, use one of the other cart operations.

Carts cannot be deleted. They expire automatically after being unused for 7 days. The lifespan of a cart restarts, however, every time a cart is modified. In this way, a cart can last for more than 7 days. If, for example, on day 6, the customer modifies a cart, the 7 day countdown starts over.

Changed in version 0.2.8: Will raise `ParameterOutOfRange` rather than `ValueError`.

`API.`**`cart_get`**(*cart_id*, *hmac*, ***params*)

The `cart_get()` operation enables you to retrieve the IDs, quantities, and prices of all of the items, including `SavedForLater` items in a remote shopping cart.

Because the contents of a cart can change for different reasons, such as availability, you should not keep a copy of a cart locally. Instead, use `cart_get()` to retrieve the items in a remote shopping cart.

To retrieve the items in a cart, you must specify the cart using the `CartId` and `HMAC` values, which are returned in the `cart_create()` operation. A value similar to `HMAC`, `URLEncodedHMAC`, is also returned. This value is the URL encoded version of the `HMAC`. This encoding is necessary because some characters, such as + and /, cannot be included in a URL. Rather than encoding the `HMAC` yourself, use the `URLEncodedHMAC` value for the HMAC parameter.

`cart_get()` does not work after the customer has used the `PurchaseURL` to either purchase the items or merge them with the items in their Amazon cart.

If the associated `cart_create()` request specified an `AssociateTag`, all `cart_get()` requests must also include a value for `AssociateTag` otherwise the request will fail.

API.**cart_add**(*cart_id*, *hmac*, *items*, ***params*)

The `cart_add()` operation enables you to add items to an existing remote shopping cart. `cart_add()` can only be used to place a new item in a shopping cart. It cannot be used to increase the quantity of an item already in the cart. If you would like to increase the quantity of an item that is already in the cart, you must use the `cart_modify()` operation.

You add an item to a cart by specifying the item's `OfferListingId`, or `ASIN` and `ListItemId`. Once in a cart, an item can only be identified by its `CartItemId`. That is, an item in a cart cannot be accessed by its ASIN or `OfferListingId`. `CartItemId` is returned by `cart_create()`, `cart_get()`, and `cart_add()`.

To add items to a cart, you must specify the cart using the `CartId` and `HMAC` values, which are returned by the `cart_create()` operation.

If the associated `cart_create()` request specified an AssociateTag, all `cart_add()` requests must also include a value for Associate Tag otherwise the request will fail.

Note: Some manufacturers have a minimum advertised price (MAP) that can be displayed on Amazon's retail web site. In these cases, when performing a Cart operation, the MAP Is returned instead of the actual price. The only way to see the actual price is to add the item to a remote shopping cart and follow the PurchaseURL. The actual price will be the MAP or lower.

Changed in version 0.2.8: Will raise `ParameterOutOfRange` rather than `ValueError`.

API.**cart_modify**(*cart_id*, *hmac*, *item_ids*, ***params*)

The `cart_modify()` operation enables you to change the quantity of items that are already in a remote shopping cart and move items from the active area of a cart to the `SaveForLater` area or the reverse.

To modify the number of items in a cart, you must specify the cart using the `CartId` and HMAC values that are returned in the `cart_create()` operation. A value similar to HMAC, `URLEncodedHMAC`, is also returned. This value is the URL encoded version of the HMAC. This encoding is necessary because some characters, such as + and /, cannot be included in a URL. Rather than encoding the HMAC yourself, use the `URLEncodedHMAC` value for the HMAC parameter.

You can use `cart_modify()` to modify the number of items in a remote shopping cart by setting the value of the Quantity parameter appropriately. You can eliminate an item from a cart by setting the value of the Quantity parameter to zero. Or, you can double the number of a particular item in the cart by doubling its Quantity. You cannot, however, use `cart_modify()` to add new items to a cart.

If the associated `cart_create()` request specified an AssociateTag, all `cart_modify()` requests must also include a value for Associate Tag otherwise the request will fail.

Changed in version 0.2.8: Will raise `ParameterOutOfRange` or `MissingParameters` rather than `ValueError`.

API.**cart_clear**(*cart_id*, *hmac*, ***params*)

The `cart_clear()` operation enables you to remove all of the items in a remote shopping cart, including `SavedForLater` items. To remove only some of the items in a cart or to reduce the quantity of one or more items, use `cart_modify()`.

To delete all of the items from a remote shopping cart, you must specify the cart using the `CartId` and `HMAC` values, which are returned by the `cart_create()` operation. A value similar to the `HMAC`, `URLEncodedHMAC`, is also returned. This value is the URL encoded version of the `HMAC`. This encoding is necessary because some characters, such as + and /, cannot be included in a URL. Rather than encoding the `HMAC` yourself, use the U''RLEncodedHMAC'' value for the HMAC parameter.

`cart_clear()` does not work after the customer has used the `PurchaseURL` to either purchase the items or merge them with the items in their Amazon cart.

Carts exist even though they have been emptied. The lifespan of a cart is 7 days since the last time it was acted upon. For example, if a cart created 6 days ago is modified, the cart lifespan is reset to 7 days.

3.3 Common request parameters

There are a number of *optional* keyword parameters which you can use to any of the afore mentioned operations.

`ContentType` Specifies the format of the content in the response. Generally, `ContentType` should only be changed for REST requests when the `Style` parameter is set to an XSLT stylesheet. For example, to transform your Product Advertising API response into HTML, set `ContentType` to `text/html`. See `Style`.

Valid Value: `text/xml` (default), `text/html`

`MarketplaceDomain` Specifies the Marketplace Domain where the request will be directed. For more information, see http://docs.amazonwebservices.com/AWSECommerceService/latest/DG/index.html?MarketplaceDomainParameter.html

`MerchantId` An optional parameter that can be used to filter search results and offer listings to only include items sold by Amazon. By default, the API will return items sold by various merchants including Amazon.

`Style` Controls the format of the data returned in Product Advertising API responses. `Style` only pertains to REST requests. Set this parameter to `XML` (default), to generate a pure XML response. Set this parameter to the URL of an XSLT stylesheet to have Product Advertising API transform the XML response. See `ContentType`.

Valid Values: URL of an XSLT stylesheet

`Validate` Prevents an operation from executing. Set the `Validate` parameter to `True` to test your request without actually executing it. When present, `Validate` must equal `True`; the default value is `False`. If a request is not actually executed (`Validate=True`), only a subset of the errors for a request may be returned because some errors (for example, `NoExactMatchesFound`) are only generated during the execution of a request.

Valid Values: `True`, `False` (default)

`Version` The version of the Product Advertising API software and WSDL to use. By default, the `2005-10-05` version is used. Alternately, specify a software version, such as `2011-08-01`. For a list of valid version numbers, refer to the Product Advertising API Release Notes. Note that the latest version of Product Advertising API is not used by default.

Valid Values: Valid WSDL version date, for example, `2011-08-01`. Default: `2005-10-05`

Note: If you want to adjust your `Version` more easily, have a look at *Using a different API version*.

`XMLEscaping` Specifies whether responses are XML-encoded in a single pass or a double pass. By default, `XMLEscaping` is `Single`, and Product Advertising API responses are encoded only once in XML. For example, if the response data includes an ampersand character (&), the character is returned in its regular XML encoding (&). If `XMLEscaping` is `Double`, the same ampersand character is XML-encoded twice (&). The `Double` value for `XMLEscaping` is useful in some clients, such as PHP, that do not decode text within XML elements.

Valid Values: `Single` (default), `Double`

Please refer to http://docs.amazonwebservices.com/AWSECommerceService/latest/DG/index.html?CommonRequestParameters.html for an up-to-date list of parameters.

CHAPTER 4

Result processing

By default this module uses lxml.objectify to parse all XML responses it receives from Amazon. However, this will only work if `lxml` is actually installed.

On some systems like Google App Engine lxml cannot be installed. Therefore there are a number of fallbacks which will be tried in the following order:

- `amazonproduct.processors.objectify.Processor`
- `amazonproduct.processors.etree.Processor`

There is also a processor using `minidom`.

- `amazonproduct.processors.minidom.Processor`

Note: If you want to use your own parser have a look at `amazonproduct.processors.BaseProcessor` and *`amazonproduct.processors.BaseResultPaginator`*

CHAPTER 5

Result pagination

New in version 0.2.5.

Changed in version 0.2.6.

One of the main advantages of this wrapper is that it provides automatic pagination of results. Rather than having to make 10 calls to get all available pages, you can simply iterate over the paginator instance that some operations return.

Example:

```
>>> api = API(locale='de')
>>> results = api.item_search('Books',
...     Publisher='Galileo Press', Sort='salesrank')
>>> results
<amazonproduct.processors._lxml.SearchPaginator object at 0x253af10>
```

The result is a `SearchPaginator` instance, which can be queried

```
>>> results.results
286
>>> results.pages
29
```

and iterated over

```
>>> for item in results:
...     print repr(item)
...
<Element {http://webservices.amazon.com/AWSECommerceService/2011-08-01}Item at 0x25441e0>
<Element {http://webservices.amazon.com/AWSECommerceService/2011-08-01}Item at 0x25442d0>
<Element {http://webservices.amazon.com/AWSECommerceService/2011-08-01}Item at 0x2544550>
<Element {http://webservices.amazon.com/AWSECommerceService/2011-08-01}Item at 0x2544500>
...
```

New pages are loaded from Amazon (up to a maximum of 10 pages) as they are required.

class `amazonproduct.processors.`**`BaseResultPaginator`**(*fun*, **args*, ***kwargs*)

Wrapper class for paginated results. This class will call the passed function iteratively until either the specified limit is reached or all result pages, which can be retrieved, are fetched.

Note: Amazon does put a rather restrictive limit on pagination. Don't expect to be able to retrieve all result pages!

A result paginator has the following attributes:

pages (same as `len(<paginator>)`) Number of *total* pages. This may differ from the number of pages actually iterated over because of limits either imposed by Amazon or yourself (using `limit`).

results Number of total results. This may differ from the number of results actually retrievable because Amazon generally limits pagination to ten pages.

current Number of result page retrieved last.

CHAPTER 6

Error handling

The most basic error is `AWSError`, which has attributes `code` and `message`. Almost all operations raise specialised exceptions.

exception `amazonproduct.errors.`**`AWSError`**(**args*, ***kwargs*)
Generic AWS error message with the following attributes:

`code` The Amazon error code (e.g. `AWS.InvalidEnumeratedParameter`)

`msg` The original error message from Amazon

`xml` XML returned from Amazon as processed by the API's result processor

You can (and should) still pass additional arguments to derived exceptions which (as with `BaseException`) will be stored in `args`.

Sometimes you may still want to access the original response. An example:

```
try:
    result = api.item_lookup(
        ['644209004461', '009800895250', '301357583001'], IdType='UPC')
except InvalidParameterValue, e:
    print 'There was an invalid ItemId!' # '301357583001'
    result = e.xml
```

Although UPC `301357583001` will cause an error to be raised, you can retrieve the parsed response (here `result` is simply replaced with `e.xml`) and continue working on it as if nothing has happened.

6.1 Occurring exceptions

exception `amazonproduct.errors.`**`CartInfoMismatch`**(**args*, ***kwargs*)
Your request contains an invalid AssociateTag, CartId and HMAC combination. Please verify the AssociateTag, CartId, HMAC and retry.

Remember that all Cart operations must pass in the CartId and HMAC that were returned to you during the CartCreate operation.

exception `amazonproduct.errors.`**`DeprecatedOperation`**(**args*, ***kwargs*)
The specified feature (operation) is deprecated.

exception `amazonproduct.errors.`**`InternalError`**(**args*, ***kwargs*)
Amazon encountered an internal error. Please try again.

exception `amazonproduct.errors.`**`InvalidCartId`**(**args*, ***kwargs*)
Your request contains an invalid value for CartId. Please check your CartId and retry your request.

exception amazonproduct.errors.**InvalidCartItem**(**args*, ***kwargs*)
The item you specified, ???, is not eligible to be added to the cart. Check the item's availability to make sure it is available.

exception amazonproduct.errors.**InvalidClientTokenId**(**args*, ***kwargs*)
The AWS Access Key Id you provided does not exist in Amazon's records.

exception amazonproduct.errors.**InvalidListType**(**args*, ***kwargs*)
The value you specified for ListType is invalid. Valid values include: BabyRegistry, Listmania, WeddingRegistry, WishList.

exception amazonproduct.errors.**InvalidOperation**(**args*, ***kwargs*)
The specified feature (operation) is invalid.

exception amazonproduct.errors.**InvalidParameterCombination**(**args*, ***kwargs*)
Your request contained a restricted parameter combination.

exception amazonproduct.errors.**InvalidParameterValue**(**args*, ***kwargs*)
The specified ItemId parameter is invalid. Please change this value and retry your request.

exception amazonproduct.errors.**InvalidResponseGroup**(**args*, ***kwargs*)
The specified ResponseGroup parameter is invalid. Valid response groups for ItemLookup requests include:

Accessories, AlternateVersions, BrowseNodes, Collections, EditorialReview, Images, ItemAttributes, ItemIds, Large, ListmaniaLists, Medium, MerchantItemAttributes, OfferFull, OfferListings, OfferSummary, Offers, PromotionDetails, PromotionSummary, PromotionalTag, RelatedItems, Request, Reviews, SalesRank, SearchBins, SearchInside, ShippingCharges, Similarities, Small, Subjects, Tags, TagsSummary, Tracks, VariationImages, VariationMatrix, VariationMinimum, VariationOffers, VariationSummary, Variations.

exception amazonproduct.errors.**InvalidSearchIndex**(**args*, ***kwargs*)
The value specified for SearchIndex is invalid. Valid values include:

All, Apparel, Automotive, Baby, Beauty, Blended, Books, Classical, DVD, Electronics, ForeignBooks, HealthPersonalCare, HomeGarden, HomeImprovement, Jewelry, Kitchen, Magazines, MP3Downloads, Music, MusicTracks, OfficeProducts, OutdoorLiving, PCHardware, Photo, Shoes, Software, SoftwareVideoGames, SportingGoods, Tools, Toys, VHS, Video, VideoGames, Watches

exception amazonproduct.errors.**ItemAlreadyInCart**(**args*, ***kwargs*)
The item you specified, ???, is already in your cart.

Deprecated since version 0.2.6.

exception amazonproduct.errors.**MissingClientTokenId**(**args*, ***kwargs*)
Request must contain AWSAccessKeyId or X.509 certificate.

exception amazonproduct.errors.**MissingParameters**(**args*, ***kwargs*)
Your request is missing required parameters. Required parameters include XXX.

exception amazonproduct.errors.**NoExactMatchesFound**(**args*, ***kwargs*)
We did not find any matches for your request.

exception amazonproduct.errors.**NoSimilarityForASIN**(**args*, ***kwargs*)
When you specify multiple items, it is possible for there to be no intersection of similar items.

exception amazonproduct.errors.**NotEnoughParameters**(**args*, ***kwargs*)
Your request should have at least one parameter which you did not submit.

exception amazonproduct.errors.**TooManyRequests**(**args*, ***kwargs*)
You are submitting requests too quickly and your requests are being throttled. If this is the case, you need to slow your request rate to one request per second.

exception amazonproduct.errors.**UnknownLocale**(**args*, ***kwargs*)
Raised when unknown locale is specified.

CHAPTER 7

Configuration

New in version 0.2.6.

There is a growing list of configuration options for the library, many of which can be passed directly to the API constructor at initialisation. Some options, such as credentials, can also be read from environment variables (e.g. `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`).

7.1 Using files

To use a config file, pass its path to the API:

```
import amazonproduct
api = amazonproduct.API(cfg='~/my-config-file')
```

If no path was specified, the API looks for configuration files in the following locations and in the following order:

- `/etc/amazon-product-api.cfg` for site-wide settings that all users on this machine will use
- `~/.amazon-product-api` for user-specific settings

The options are merged into a single, in-memory configuration that is available.

The following sections and options are currently recognized within the config file.

`Credentials` The Credentials section is used to specify the AWS credentials used for all requests.

`access_key` Your AWS access key

`secret_key` Your AWS secret access key

`associate_tag` Your AWS associate ID

Example:

```
[Credentials]
access_key = <your access key>
secret_key = <your secret key>
associate_tag = <your associate id>
```

Note: Stating the obvious: Your access key is *not* `<your access key>` but something like `10RZZJBK6YBQASX213G2`.

7.2 Using config dict

If you need to configure the API at runtime you can also pass the config values as dict:

```
import amazonproduct
config = {
    'access_key': 'ABCDEFG1234X',
    'secret_key': 'Ydjkei78HdkffdklieAHDJWE3134',
    'associate_tag': 'redtoad-10',
    'locale': 'us'
}
api = amazonproduct.API(cfg=config)
```

7.3 Environment variables

You can also set the following environment variables:

AWS_ACCESS_KEY_ID Your AWS access key

AWS_SECRET_ACCESS_KEY Your AWS secret access key

AWS_ASSOCIATE_TAG Your AWS associate ID

AWS_LOCALE Your API locale

Important: Environment variables will always take precedence over values from config files *but not from config dict*!

7.4 Order of precedence

- Parameters specified by environment variables
- User-specific parameters from `~/.amazon-product-api`

The following table gives an overview which values can be defined where:

config file	environment variable
access_key	AWS_ACCESS_KEY_ID
secret_key	AWS_SECRET_ACCESS_KEY
associate_tag	AWS_ASSOCIATE_TAG
locale	AWS_LOCALE

CHAPTER 8

More advanced uses

8.1 Using a different API version

Amazon releases a new API version every once in a while in order to add or change features and operations. Usually, you won't have to worry about this because with each release of this wrapper the latest API version will be used by default.

If you do want to change the API version used, however, you can simply specify which one you like:

```
api = API(...)
api.VERSION = '2010-10-01'
```

Warning: As of Feb 21st, 2012 all API versions prior to 2011-08-01 will no longer be supported!

8.2 Use your own XML parsing library

New in version 0.2.3.

You don't need to use `lxml.objectify`. A custom response processor can be defined using any mechanism you like. For instance, here is one using `xml.minidom`:

```
import xml.dom.minidom
def minidom_response_parser(fp):
    root = xml.dom.minidom.parse(fp)
    # parse errors
    for error in root.getElementsByTagName('Error'):
        code = error.getElementsByTagName('Code')[0].firstChild.nodeValue
        msg = error.getElementsByTagName('Message')[0].firstChild.nodeValue
            raise AWSError(code, msg)
        return root

# Now let's use this instead of the default one
api = API(AWS_KEY, SECRET_KEY, 'uk', processor=minidom_response_parser)
root = api.item_lookup('0718155157')
print root.toprettyxml()
# ...
```

Note: Make sure your response parser raises an `AWSError` with the appropriate error code and message.

8.3 Caching responses

New in version 0.2.5.

Sometimes when developing or when it is foreseeable that the very same request will be sent over and over again, it might be better to cache API responses from Amazon for a short time in order to avoid going over you request limit.

CHAPTER 9

Developer FAQ

Here is a growing collection of questions that pop up regularly.

9.1 I read the API docs but I can't manage to get this to work

The XML structure returned by Amazon is sometimes not easy to understand. Try the following:

```
from lxml import etree
api = API(locale='...')
results = api.call(Operation='...') # your API call
print etree.tostring(results, pretty_print=True)
```

It will print the XML response nicely formatted.

9.2 Which locale should I use and why is this important?

Amazon is a world-wide venture. Product Advertising API is as well. Product Advertising API operates in six locales:

- CA (Canada)
- CN (China)
- DE (Germany)
- ES (Spain)
- FR (France)
- IT (Italy)
- JP (Japan)
- UK (United Kingdom)
- US (United States of Amerika)

Each of these locales is serviced by an Amazon web site that uses the local language, local customs, and local formatting. For example, when you look at the DE homepage for Amazon, you see the listings in German. If you purchased an item, you would find the price in Euros, and, if you were to purchase a movie, you would find that the movie rating would conform to the movie rating system used in Germany.

Product Advertising API responses contain the same localized information. The correct locale is determined by examining the endpoint in the request.

9.3 Can I use this wrapper on Google App Engine (GAE)?

This wrapper relies by default on lxml.objectify to parse the returned XML responses from Amazon which is built with libxml, a C library. And this will not work on GAE.

For the time being there is no solution that will work out of the box. You can, however, use a different XML parser (see *Use your own XML parsing library*)!

9.4 I keep getting `InvalidParameterValue` errors. What am I doing wrong?

The Amazon webservice returns an `InvalidParameterValue` error if you enter a *wrong* ISBN. Wrong, as it seems, can mean the format is wrong (too short) or contains invalid characters (e.g. dashes "-").

Surprisingly, wrong can even mean that you used the *wrong locale*! For instance, you cannot retrieve data for an English book (ISBN 9780596158064) from locale `de` or for a German book (ISBN 9783836214063) from locale `us` - but using locale `uk` works for both!

Try your query again using a valid ISBN and play around with the locale. You can set the locale at initialisation:

```
from amazonproduct import API
AWS_KEY = '...'
SECRET_KEY = '...'
api = API(AWS_KEY, SECRET_KEY, "uk")
root = api.item_lookup('9783836214063', IdType='ISBN', SearchIndex='Books')
```

9.5 Why yet another implementation?

There are a number of alternatives available:

- PyAmazon, originally written by Mark Pilgrim, then taken over by Michael Josephson. Development seems to have stalled, with the last release in August 2004.
- Kung Xi's pyaws forked pyamazon to support the then most recent Amazon Web Service and give developers more control of the incoming data. Sometime after version 0.2.0, development over at sourceforge was dropped without warning and continued at http://trac2.assembla.com/pyaws with version 0.3.0, which was released in May 2008.

  This module seems to be the most widely used. It hasn't been updated however in quite some time. A fork of this project is maintained here.
- In October 2008 David Jane started pyecs after stumbling accross pyamazon. He decided that "a new, more class and iterator-oriented approach would be better." However, it only supports a subset. Last commit was in November 2008.
- There is a clever hack using boto to create the URL, although this library is originally designed to allow communication with Amazon's cloud APIs.

So why write your own then? First and foremost, since August 15, 2009 all calls to Amazon's Product Advertising API must be authenticated using request signatures. The afore mentioned libraries did not support this out of the box at the time. And yes... writing something from scratch is always more appealing.

More recently I stumbled across another alternative:

- Dan Loewenherz's bottlenose makes sending requests to Amazon as easy as

```
import bottlenose
amazon = bottlenose.Amazon("access_key_id", "secret_access_key")
response = amazon.ItemSearch(ItemId="0596520999", ResponseGroup="Images",
    SearchIndex="Books", IdType="ISBN")
```

It has a straight-forward API, is easy to use and supports all operations out of the box. You only have to take care of processing the response. I must steal some ideas from this module!

9.6 I found a bug! What do I do now?

You can do two things:

1. File a bug report (but please look at the list of know issues before)
2. Send an e-mail to the mailing list.

Any feedback is welcome!

CHAPTER 10

How to contribute

Development happens at http://bitbucket.org/basti/python-amazon-product-api.

Contributions are always welcome. You can do this by

- filing bug reports,
- discussing new ideas on the mailing list or
- sending me patches.

If you do the latter, please make sure that all the tests run successfully (see also *Running the Tests*).

10.1 Setting up a development environment

What you will need to work on this module:

- lxml
- pytest (>2.0)
- pytest-localserver
- Sphinx
- tox (optional)

It might be a good idea to install all of the above mentioned dependencies into a virtualenv (I prefer to use virtualenvwrapper).

10.2 Running the Tests

There are a large number of tests to check for inter-version and inter-locale consistencies. The simplest way of running them is to run

```
python setup.py test
```

in the root directory. The tests require pytest. In order to check all supported Python versions (currently 2.4 - 2.7), I use tox.

When adding new tests, you need to pass your credentials to the API. Have a look at *Configuration* to see how to set it up. Your credentials will *not be stored* in any files!

Note: Providing tests with your pull request will increase the chances of your changes being accepted by a factor of

one gazillion!

CHAPTER 11

Changes

11.1 0.2.8 (2014-03-30)

- Fixed #31: Using generic error factory *_e()*, the original parsed XML can be accessed in case of failure.
- Added *InvalidAccount* and *InvalidSignature* exceptions (thanks to Jannis Gebauer)

11.2 0.2.7 (2013-10-08)

Small bugfix release!

- Examples fixed. Processors can now be processor instances again.

11.3 0.2.6 (2013-09-14) "Humperdinck"

- Supports API version 2011-08-01
- Config files added
- Almost total rewrite of processors backend. Will work now with elementtree from stdlib, too.
- #26: Added endpoints for CN, ES and IT.
- Added RetryAPI to contrib package thanks to Jerry Ji.
- Documentation was overhauled.
- As of 2012-10-01 there are no more XSLT endpoints!

11.4 0.2.5 (2011-09-19) "Buttercup"

- Support for XSLT requests.
- Support for Associate tags thanks to Kilian Valkhof.
- New API versions 2010-12-01, 2010-11-01, 2010-10-01, 2010-09-01 and 2010-08-06 added.
- Fixed #16: Cannot install module under Python 2.4 without pycrypto being installed first.
- tox (and hudson) are now used for testing all supported Python versions (which includes Python 2.7 now, too).

- Test server is replaced with pytest-localserver.
- Fixed #18: Throttling no longer block CPU (Thanks to Benoit C).
- Added response-caching API (in `amazonproduct.contrib.caching`) to ease development (Thanks to Dmitry Chaplinsky for the idea).
- API explicitly warns about deprecated operations.

Important: The following operations are deprecated since 15 July 2010 and are now answered with a '410 Gone' (and a `DeprecatedOperation` exception):

- CustomerContentLookup
- CustomerContentSearch
- Help
- ListLookup
- ListSearch
- TagLookup
- TransactionLookup
- VehiclePartLookup
- VehiclePartSearch
- VehicleSearch

- Added new exceptions InvalidClientTokenId and MissingClientTokenId.
- `REQUESTS_PER_SECONDS` can now be floats as well (e.g. 2500/3600.0).
- Added test options options `--api-version`, `--locale` and `--refetch`.

11.5 0.2.4.1 (2010-06-23)

Bugfix release! High time I get some continuous integration set up!

- Fixed #13: The module did not run under Python 2.4. Ooops!

11.6 0.2.4 (2010-06-13)

- Locale parameter is now required at initialisation.

```
# before you could write
api = API(AWS_KEY, SECRET_KEY)

# now you have to specify your locale
api = API(AWS_KEY, SECRET_KEY, 'de')
```

- Custom test server (`tests.server.TestServer`) added. It runs on localhost and mimicks the Amazon webservice by replaying local XML files.
- Testing now supports multiple locales. Please not that you have to run `python setup.py test` to run the unittests.

- ResultPaginator now also works with XPath expressions for attributes (Bug reported Giacomo Lacava).
- Custom lookup for XML elements (during parsing) ensures that `<ItemId/>` and `<ASIN>` are now always `objectify.StringElement` (Bug reported by Brian Browning).
- Fixed #11: Module can now be installed library without lxml being installed first.
- Regular expressions for parsing error messages can now deal with the Japanese version.

> **Warning:** The support for the Japanese locale (`jp`) is still very experimental! A few error messages have still to be translated and the functionality has to be confirmed. If you know Japanese, get in touch!

11.7 0.2.3 (2010-03-20)

- Tests run now for all API versions. Test cases can now be told which versions to use (class attribute `api_versions` set to i.e. `['2009-10-01']`).
- A custom AWS response processor can now be defined. For instance, here is one using `xml.minidom` instead of `lxml`:

```
    def minidom_response_parser(fp):
        root = parse(fp)
        # parse errors
        for error in root.getElementsByTagName('Error'):
            code = error.getElementsByTagName('Code')[0].firstChild.nodeValue
            msg = error.getElementsByTagName('Message')[0].firstChild.nodeValue
                raise AWSError(code, msg)
            return root
        api = API(AWS_KEY, SECRET_KEY, processor=minidom_response_parser)
        root = api.item_lookup('0718155157')
        print root.toprettyxml()
        # ...
```

- Fixed #3: Support for API Version 2009-11-01.
- Fixed #4: When using a bad parameter combination, an `InvalidParameterCombination` exception is raised.
- Fixed #5: `InvalidSearchIndex` is raised when unknown `SearchIndex` is specified.
- Fixed #7: Specifying API versions works now for more than just one test per test case.
- The `setup.py` command has been empowered a bit with the following additional options: `test`, `build_sphinx`, `upload_sphinx`.
- `ResultPaginator` attributes `_get_current_page_numer`, `_get_total_results` and `_get_total_page_numer` are now private.

11.8 0.2.2 (2010-01-30)

- `browse_node_lookup` operation added.
- `help` operation added.
- `list_lookup` and `list_search` operations added.
- Default timeout for API calls is set to 5 sec.

- Test cases for correct parsing of XML responses added. Local XML files are used for testing (if available) stored in separate directories according to API version. These can be overwritten when config value `OVERWRITE_TESTS` is set to `True`.
- `InvalidItemId` exception is replaced by more general `InvalidParameterValue` exception.

11.9 0.2.1 (2009-11-20)

- Support for Python 2.4 added.
- Fixed #2: ResultPaginator now returns `None` if the XPath expression doesn't find the node it's looking for.

11.10 0.2.0 (2009-11-07) "Westley"

This is the first *public* release. We're now available via the Cheeseshop! http://pypi.python.org/pypi/python-amazon-product-api

- The module is no longer a package. Please use `import amazonproduct` (instead of `import amazon.product`) now.
- SimilarityLookup is now supported.
- Updated to support version 2009-10-01.
- Documentation added (made with http://sphinx.pocoo.org).
- New artwork.

11.11 0.1 (2009-09-30) "Fezzik"

Initial release.

CHAPTER 12

License

This module is release under the BSD License.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Amazon Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.