EVE Market Data Structures Documentation

Release 0.6

Greg Taylor

February 02, 2017

Contents

EVE Market Data Structures is a set of Python data structures for EVE Online market data. The most common uses for EMDS are EMDR consumers and market data uploaders.

License: EVE Market Data Structures is licensed under the BSD License.

These links may also be useful to you.

- Source repository: https://github.com/gtaylor/EVE-Market-Data-Structures
- Issue tracker: https://github.com/gtaylor/EVE-Market-Data-Structures/issues
- IRC Room: irc.coldfront.net #emdr
- Mailing list: https://groups.google.com/forum/#!forum/eve-emdr
- @gctaylor on Twitter: https://twitter.com/#!/gctaylor

Documentation

1.1 Installation

The only hard requirement for using EMDS is Python 2.6, though, Python 2.7 is strongly recommended. PyPy should also work just fine.

To install EMDS, you may use `pip`:

```
pip install emds
```

or `easy_install`:

```
easy_install emds
```

1.1.1 JSON modules

EMDS can run with any of the following JSON modules:

- ujson
- simplejson
- Python 2.6+ built-in json module

The first module detected in the list above (in order) will be the one that is used. ujson is the fastest, and simplejson is likely to be the best combination of being up to date and very mature.

1.2 Quickstart

Market data is comprised of *orders* and *history*. Orders represent individual buy/sell orders, and history entries are points of data on the cost and availability of a given item.

In both cases, the data structures are similar in that there is a container class (*`MarketOrderList`* and *`MarketHistoryList`*), an item+region grouper class (*`MarketItemsInRegionList`* and *`HistoryItemsInRegionList`*), and a unit of data (*`MarketOrder`* and *`MarketHistoryEntry`*).

The two top-level classes are best thought of as containers for any number of orders or history entries. *`MarketOrderList`* and *`MarketHistoryList`* contain nothing but some basic info about where the data came from, along with some convenience methods for manipulating and retrieving said data.

Beneath the top level List classes are two classes (`MarketItemsInRegionList` and `HistoryItemsInRegionList`), that group orders together based on Item ID and Region ID. For example, all orders of Type ID 12345 in Region ID 6789 are grouped together, much like they are in the in-game market browser. Within these two classes are the individual order and history instances (`MarketOrderList` and `MarketHistoryList`).

The hierarchy ends up looking like this:

- `MarketOrderList`
  - `MarketItemsInRegionList`
    - `MarketOrder`
- `MarketHistoryList`
  - `HistoryItemsInRegionList`
    - `MarketHistoryList`

1.2.1 The importance of timezone awareness

EVE Online deals with nothing but UTC times. Python's default `datetime.datetime.now()` and `datetime.datetime.utcnow()` return objects without `tzinfo` set, which means that the objects are considered "naive", with regards to what timezone they are in.

EMDS requires that all times being passed into the data structures have a timezone specified via the `tzinfo` attribute. Simple using `now()` or `utcnow()` will result in exceptions being raised, so don't do that.

If you're simply wanting to get the current time in UTC, you can use our helper function:

```
>>> from emds.common_utils import now_dtime_in_utc
>>> now_dtime_in_utc()
datetime.datetime(...)
```

If you have an existing naive datetime.datetime (has no `tzinfo` value) whose hour value is set to represent UTC, you can use pytz to "enlighten" the naive datetime instance:

```
>>> import pytz; import datetime
>>> utc = pytz.timezone("UTC")
>>> naive_utcnow = datetime.datetime.utcnow()
>>> enlightend_utcnow = naive_utcnow.replace(tzinfo=utc)
```

This does no conversion of the `hour` value, it simply replaces the `None` tzinfo value with the UTC tzinfo object. The datetime object now unambiguously says it is in UTC, whereas before, it was a naive object that just had an hour value of some sort in an un-specified timezone.

Note: You probably don't want to ever bother with datetime.datetime.now(). This returns local time, which is just not something you want to get involved with.

1.2.2 Creating/populating market order lists

Creating and populating market order lists is as simple as instantiating a `MarketOrderList` object and using its `add_order` method.

```
from emds.data_structures import MarketOrder, MarketOrderList
from emds.common_utils import now_dtime_in_utc

order_list = MarketOrderList()
order_list.add_order(MarketOrder(
    order_id=2413387906,
    is_bid=True,
    region_id=10000123,
    solar_system_id=30005316,
    station_id=60011521,
    type_id=2413387906,
    price=52875,
    volume_entered=10,
    volume_remaining=4,
    minimum_volume=1,
    order_issue_date=now_dtime_in_utc(),
    order_duration=90,
    order_range=5,
    generated_at=now_dtime_in_utc()
))
```

Behind the scenes, this is creating a *MarketItemsInRegionList* instance that will contain all orders with Type ID `2413387906` and Region ID `10000123`. Orders are grouped based on their Type and Region IDs.

1.2.3 Iterating over market order lists

Assuming you have a *MarketOrderList* instance you want to pull order data from, there are two primary ways to do so.

If you are only concerned with pulling all orders out, without caring whether certain regions+item combos are empty, simply use the *MarketOrderList.get_all_orders_ungrouped* generator:

```
order_list = MarketOrderList()
# Add your orders here.
# ...
for order in order_list.get_all_orders_ungrouped():
    # order is a MarketOrder instance.
    print order.order_id, order.type_id, order.region_id
```

If you need to know that certain item+region combinations held no orders, you'll need to iterate through the orders in all of the MarketOrderList's *MarketItemsInRegionList* instances:

```
order_list = MarketOrderList()
# Add your orders here.
# ...
for ir_group in order_list.get_all_order_groups():
    # ir_group is a MarketItemsInRegionList, which contains orders
    # You can check to see if there are any orders
    num_orders = len(ir_group)
    # If it's 0, you could mark it as such in your application.
    for order in ir_group:
        # order is a MarketOrder instance.
        print order.order_id, order.type_id, order.region_id
```

1.2.4 Creating/populating market history lists

Creating and populating market order lists is as simple as instantiating a *MarketHistoryList* object and using its *add_entry* method.

```
from emds.data_structures import MarketHistoryEntry, MarketHistoryList
from emds.common_utils import now_dtime_in_utc

history_list = MarketHistoryList()
history_list.add_entry(MarketHistoryEntry(
    type_id=2413387906,
    region_id=10000068,
    historical_date=now_dtime_in_utc(),
    num_orders=5,
    low_price=5.0,
    high_price=10.5,
    average_price=7.0,
    total_quantity=200,
    generated_at=now_dtime_in_utc(),
))
```

Behind the scenes, this is creating a *HistoryItemsInRegionList* instance that will contain all entries with Type ID `2413387906` and Region ID `10000068`. History entries are grouped based on their Type and Region IDs.

1.2.5 Iterating over market history lists

Assuming you have a *MarketHistoryList* instance you want to pull history data from, there are two primary ways to do so.

If you are only concerned with pulling all history entries out, without caring whether certain regions+item combos lack data, simply use the *MarketHistoryList.get_all_entries_ungrouped* generator:

```
history_list = MarketHistoryList()
# Add your history entries here.
# ...
for entry in history_list.get_all_entries_ungrouped():
    # entry is a MarketHistoryEntry instance.
    print entry.type_id, entry.region_id, entry.average_price
```

If you need to know that certain item+region combinations held no history data, you'll need to iterate through the entries in all of the MarketHistoryList's *HistoryItemsInRegionList* instances:

```
history_list = MarketHistoryList()
# Add your history entries here.
# ...
for ir_group in history_list.get_all_entries_grouped():
    # ir_group is a HistoryItemsInRegion, which contains history entries
    # You can check to see if there are any entries
    num_entries = len(ir_group)
    # If it's 0, you could mark it as such in your application.
    for entry in ir_group:
        # entry is a MarketHistoryEntry instance.
        print entry.type_id, entry.region_id, entry.average_price
```

1.3 Serializing to/from Unified Format

EMDS currently only supports Unified Uploader Data Interchange Format (UUDIF). While there are other “proprietary” formats that are geared towards individual market sites, the goal of EMDS is to offer a set of data structures and serializers for the community-backed formats.

In the spirit of supporting the community-drafted and backed formats (currently UUDIF), this is all we’ll support until/unless a new format comes along, is vetted by the community, and receives traction.

1.3.1 Dumping order and history lists to UUDIF

Regardless of whether you are dealing with a `MarketOrderList` or a `MarketHistoryList`, serializing to UUDIF works the same:

```
from emds.formats import unified

order_list = MarketOrderList()
# Add your orders here.
# ...

# This spits out the JSON UUDIF message string. encode_to_json() accepts
# MarketOrderList and MarketHistoryList instances.
encoded_order_list = unified.encode_to_json(order_list)
```

1.3.2 Parsing UUDIF order and history messages

Parsing a UUDIF message results in either a `MarketOrderList` or a `MarketHistoryList` instance being returned:

```
from emds.formats import unified

data = """
{
  "resultType" : "orders",
  "version" : "0.1alpha",
  "uploadKeys" : [
    { "name" : "emk", "key" : "abc" },
    { "name" : "ec" , "key" : "def" }
  ],
  "generator" : { "name" : "Yapeal", "version" : "11.335.1737" },
  "currentTime" : "2011-10-22T15:46:00+00:00",
  "columns" : ["price","volRemaining","range","orderID","volEntered","minVolume","bid","issueDate","
  "rowsets" : [
    {
      "generatedAt" : "2011-10-22T15:43:00+00:00",
      "regionID" : 10000065,
      "typeID" : 11134,
      "rows" : [
        [8999,1,32767,2363806077,1,1,false,"2011-12-03T08:10:59+00:00",90,60008692,30005038],
        [11499.99,10,32767,2363915657,10,1,false,"2011-12-03T10:53:26+00:00",90,60006970,null],
        [11500,48,32767,2363413004,50,1,false,"2011-12-02T22:44:01+00:00",90,60006967,30005039]
      ]
    },
    {
      "generatedAt" : "2011-10-22T15:42:00+00:00",
      "regionID" : null,
      "typeID" : 11135,
      "rows" : [
        [8999,1,32767,2363806077,1,1,false,"2011-12-03T08:10:59+00:00",90,60008692,30005038],
        [11499.99,10,32767,2363915657,10,1,false,"2011-12-03T10:53:26+00:00",90,60006970,null],
        [11500,48,32767,2363413004,50,1,false,"2011-12-02T22:44:01+00:00",90,60006967,30005039]
      ]
    }
  ]
}
"""
# This spits out a MarketOrderList instance that is ready to be
# iterated over.
order_list = unified.parse_from_json(data)
```

1.4 Data Structure Reference

Below is a reference to all of the important EVE market data structures. For examples on how to use them, see Quickstart.

1.4.1 MarketOrderList

class `emds.data_structures.`**`MarketOrderList`**(*upload_keys=None*, *order_generator=None*, **args*, ***kwargs*)

A list of MarketOrder objects, with some added features for assisting with serializing to the Unified Uploader Data Interchange format.

Attr list_type This may be used in your logic to separate orders from history.

`add_order`(*order*)

Adds a MarketOrder instance to the list of market orders contained within this order list. Does some behind-the-scenes magic to get it all ready for serialization.

Parameters **`order`** (`MarketOrder`) – The order to add to this order list.

`get_all_order_groups`()

Uses a generator to return all grouped Item+Region combos, in the form of `MarketItemsInRegionList` instances. This is useful in that it is possible to express a lack of an item in a specific region.

Note: This is a generator!

Return type generator

Returns Generates a list of `MarketItemsInRegionList` instances, which contain `MarketOrder` instances.

`get_all_orders_ungrouped`()

Uses a generator to return all orders within. `MarketOrder` objects are yielded directly, instead of being grouped in `MarketItemsInRegionList` instances.

Note: This is a generator!

Return type generator

Returns Generates a list of `MarketOrder` instances.

`set_empty_region`(*region_id*, *type_id*, *generated_at*, *error_if_orders_present=True*)
Prepares for the given region+item combo by instantiating a `MarketItemsInRegionList` instance, which will track region ID, type ID, and generated time. This is mostly used for the JSON deserialization process in case there are no orders for the given region+item combo.

Parameters

- **`region_id`** (*`int`*) – The region ID.
- **`type_id`** (*`int`*) – The item's type ID.
- **`generated_at`** (*`datetime.datetime`*) – The time that the order set was generated.
- **`error_if_orders_present`** (*`bool`*) – If True, raise an exception if an order already exists for this item+region combo when this is called. This failsafe may be disabled by passing False here.

1.4.2 MarketItemsInRegionList

class `emds.data_structures.`**`MarketItemsInRegionList`**(*region_id*, *type_id*, *generated_at*)
This is an intermediary container that stores MarketOrders for a particular item+region combo. The primary reason it exists is to store generation times for item+region combos that lack orders, since we can't just yank from the first order's time in that case.

Attr orders A list of MarketOrder objects.

`add_order`(*order*)
Adds a `MarketOrder` instance to this region+item list.

Parameters **`order`** (`MarketOrder`) – The order to add.

1.4.3 MarketOrder

class `emds.data_structures.`**`MarketOrder`**(*order_id*, *is_bid*, *region_id*, *solar_system_id*, *station_id*, *type_id*, *price*, *volume_entered*, *volume_remaining*, *minimum_volume*, *order_issue_date*, *order_duration*, *order_range*, *generated_at*)
Represents a market buy or sell order.

1.4.4 MarketHistoryList

class `emds.data_structures.`**`MarketHistoryList`**(*upload_keys=None*, *history_generator=None*, **args*, ***kwargs*)
A class for storing market order history for serialization.

Attr list_type This may be used in your logic to separate orders from history.

`add_entry`(*entry*)
Adds a MarketHistoryEntry instance to the list of market history entries contained within this instance. Does some behind-the-scenes magic to get it all ready for serialization.

Parameters **`entry`** (`MarketHistoryEntry`) – The history entry to add to instance.

get_all_entries_grouped()

Uses a generator to return all grouped Item+Region combos, in the form of `HistoryItemsInRegion` instances. This is useful in that it is possible to express a lack of an item in a specific region.

Note: This is a generator!

Return type generator

Returns Generates a list of `HistoryItemsInRegion` instances, which contain `MarketHistoryEntry` instances.

get_all_entries_ungrouped()

Uses a generator to return all history entries within. `MarketHistoryEntry` objects are yielded directly, instead of being grouped in `HistoryItemsInRegion` instances.

Note: This is a generator!

Return type generator

Returns Generates a list of `MarketHistoryEntry` instances.

set_empty_region(*region_id*, *type_id*, *generated_at*, *error_if_entries_present=True*)

Prepares for the given region+item combo by instantiating a `HistoryItemsInRegionList` instance, which will track region ID, type ID, and generated time. This is mostly used for the JSON deserialization process in case there are no orders for the given region+item combo.

Parameters

- **region_id** (*int*) – The region ID.
- **type_id** (*int*) – The item's type ID.
- **generated_at** (*datetime.datetime*) – The time that the order set was generated.
- **error_if_entries_present** (*bool*) – If True, raise an exception if an entry already exists for this item+region combo when this is called. This failsafe may be disabled by passing False here.

1.4.5 HistoryItemsInRegionList

class `emds.data_structures.`**HistoryItemsInRegionList**(*region_id*, *type_id*, *generated_at*)

This is an intermediary container that stores MarketHistoryEntry for a particular item+region combo. The primary reason it exists is to store generation times for item+region combos that lack orders, since we can't just yank from the first order's time in that case.

Attr orders A list of MarketHistoryEntry objects.

add_entry(*entry*)

Adds a `MarketHistoryEntry` instance to this region+item list.

Parameters **entry** (*MarketHistoryEntry*) – The history entry to add to instance.

1.4.6 MarketHistoryEntry

class `emds.data_structures.`**`MarketHistoryEntry`**(*type_id*, *region_id*, *historical_date*, *num_orders*, *low_price*, *high_price*, *average_price*, *total_quantity*, *generated_at*)

Represents a single point of market history data.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search