Bookied Documentation

Release 0.0.1

Fabian Schuh

Feb 24, 2021

Contents

BookieSports is a module that contains the management information for BOS. This management information describes which sports are supported, which leagues and participants are available and how and what betting markets are created and resolved.

CHAPTER 1

Structure

This repository contains

- the sports with emta data supported by bookied
- schema files for validation of the provided data
- a python module to facilitate loading of the data

Folders: * *bookiesports/*: Contains the module that can be loaded from python to obtain the sports data. * *bookiesports/bookiesports*: Each sport has it's own folder which carries the most important information in a sports-specific *index.yaml* file. * *bookiesports/schema/*: Contains the yaml formated json schemata for validation of the bookie sports files.

CHAPTER 2

Outline

2.1 Installation

```
pip3 install bookiesports
```

2.2 bookiesports package

2.2.1 Submodules

bookiesports.cli module

bookiesports.datestring module

bookiesports.datestring.**date_to_string**(*date_object=None*)
rfc3339 conform string represenation of a date can also be given as str YYYY-mm-dd HH:MM:SS

bookiesports.datestring.**string_to_date**(*date_string=None*)
assumes rfc3339 conform string and creates date object

bookiesports.exceptions module

exception bookiesports.exceptions.**SportsNotFoundError**
Bases: Exception

bookiesports.log module

bookiesports.normalize module

exception bookiesports.normalize.**EventGroupNotNormalizableException**
Bases: *bookiesports.normalize.NotNormalizableException*

class bookiesports.normalize.**IncidentsNormalizer**(*chain=None*)
Bases: object

This class serves as the normalization entry point for incidents. All events / event group and participant names are replaced with the counterpart stored in the bookiesports package.

DEFAULT_CHAIN = 'beatrice'
default chosen chain for bookiesports

NOT_FOUND = {}
As class variable to have one stream for missing normalization entries

NOT_FOUND_FILE = None
If normalization errors should be written to file, set file here

normalize(*incident*, *errorIfNotFound=False*)

static not_found(*key*)

static use_chain(*chain*, *not_found_file=None*)

exception bookiesports.normalize.**NotNormalizableException**
Bases: Exception

exception bookiesports.normalize.**ParicipantNotNormalizableException**
Bases: *bookiesports.normalize.NotNormalizableException*

exception bookiesports.normalize.**SportNotNormalizableException**
Bases: *bookiesports.normalize.NotNormalizableException*

2.2.2 Module contents

class bookiesports.**BookieSports**(*chain=None*, *override_cache=False*, ***kwargs*)
Bases: dict

This class allows to read the data provided by bookiesports

On instantiation of this class the following procedure happens internally:

1. Open the directory that stores the sports
2. Load all Sports
3. **For each sport, load the corresponding data subset (event** groups, events, rules, participants, etc.)
4. Validate each data subset
5. Perform consistency checks
6. Instantiate a dictionary (self)

As a result, the following call will return a dictionary with all the bookiesports:

```
from bookiesports import BookieSports
x = BookieSports()
```

Parameters

- **chain** (*string*) – One out 'alice', 'beatrice', or 'charlie' to identify which network we are working with. Can also be a relative path to a locally stored copy of a sports folder
- **override_cache** (*string*) – if true, cache is ignored and sports folder is forcibly reloaded and put into cache
- **network** (*string*) – deprecated, please use chain

It is possible to overload a custom sports_folder by providing it to `BookieSports` as parameter.

BASE_FOLDER = '/home/docs/checkouts/readthedocs.org/user_builds/bookiesports/checkouts

CHAIN_CACHE = {}
Singelton to store data and prevent rereading if BookieSports is instantiated multiple times

DEFAULT_CHAIN = 'beatrice'

JSON_SCHEMA = None
Schema for validation of the data

SPORTS_FOLDER = None

chain_id

static list_chains()

static list_networks()
@deprecated please use list_chains

network
@deprecated use self.index

network_name
@deprecated please use self.chain

static version()

2.3 Schema

For validation of the data format presented in the sports folder, a validation is performed. The corresponding validation schemata are stored in the `schema/` subdirectory and used internally when instantiating *bookiesports.BookieSports*.

2.3.1 Schemata

Genera definitions

```
definitions:

 identifier:
  type: string
  description: Identification string for the

 id:
  description: Blockchain id of the object (e.g. 1.16.0)
  pattern: "^[0-9]*\\.[0-9]*\\.[0-9]*$"
```

(continues on next page)

(continued from previous page)

```
 internationalized_name:
  type: object
  description: Internationalized name
  properties:
   en:
    type: string
    description: English name of the sport
  required:
   - en

 aliases:
  type:
   - "null"
   - array
  oneOf:
   - type: "null"
   - type: array
     description: List of known aliases
     items:
      type: string

 asset:
  type: array
  description: Asset symbol
  uniqueItems: true
  items:
   - type: string
   - enum:
      - PPY
      - BTC
      - BTCTEST
      - BTF
      - BTFUN
      - TEST
```

Sport

```
$schema: "http://json-schema.org/draft-06/schema#"
title: BookieSports::Sport
description: Format for BookieSports::Sport
type: object
properties:

 identifier:
  $ref: "#/definitions/identifier"

 name:
  $ref: "#/definitions/internationalized_name"

 aliases:
  $ref: "#/definitions/aliases"

 id:
  $ref: "#/definitions/id"
```

(continues on next page)

(continued from previous page)

```
 eventgroups:
  type: array
  description: list of event groups that are in this sport
  items:
   type: string

required:
 - identifier
 - name
 - id
 - eventgroups
```

Eventgroup

```
$schema: "http://json-schema.org/draft-04/schema#"
title: BookieSports::EventGroup
description: Format for BookieSports::EventGroup
type: object
properties:

 identifier:
  $ref: "#/definitions/identifier"

 name:
  $ref: "#/definitions/internationalized_name"

 aliases:
  $ref: "#/definitions/aliases"

 id:
  $ref: "#/definitions/id"

 participants:
  description: Identifier for the teams
  type: string

 bettingmarketgroups:
  type: array
  description: list of identifiers for the betting market groups
  items:
   type: string

 eventscheme:
  type: object
  description: Internationalized name after which the events are named on creation
  properties:
   name:
    $ref: "#/definitions/internationalized_name"

 start_date:
  type: string
  format: date-time

 finish_date:
  type: string
```

(continues on next page)

(continued from previous page)

```
  format: date-time

 leadtime_Max:
  type: number

required:
 - identifier
 - name
 - id
 - participants
 - bettingmarketgroups
 - eventscheme
   #- start_date
   #- finish_date
   #- leadtime_Max
```

Participant

```
$schema: "http://json-schema.org/draft-06/schema#"
title: BookieSports::MarketBettingGroup
description: Format for BookieSports::MarketBettingGroup
type: object
properties:

 participants:
  description: List of participants
  type: array
  items:
   type: object
   properties:

    identifier:
     $ref: "#/definitions/identifier"

    aliases:
     $ref: "#/definitions/aliases"

    name:
     $ref: "#/definitions/internationalized_name"

required:
 - participants
```

Rule

```
$schema: "http://json-schema.org/draft-06/schema#"
title: BookieSports::MarketBettingGroup
description: Format for BookieSports::MarketBettingGroup
type: object
properties:

 identifier:
  $ref: "#/definitions/identifier"
```

(continues on next page)

(continued from previous page)

```
 name:
  $ref: "#/definitions/internationalized_name"

 description:
  $ref: "#/definitions/internationalized_name"

 id:
  $ref: "#/definitions/id"

 grading:
  type: object
  description: Grading for the rule
  properties:

   metric:
    type: string
    description: Calculate metric according to this

   resolutions:
    type: array
    descriptions: Resolve betting markets according to the rules here
    items:
     type: object
     properties:
      win:
       type: string
       description: If true this market is win
      not_win:
       type: string
       description: If true this market is not_win
      void:
       type: string
       description: If true this market is void

  required:
   - metric
   - resolutions

required:
 - identifier
 - id
 - name
 - description
 - grading
```

BettingMarketGroup

```
$schema: "http://json-schema.org/draft-06/schema#"
title: BookieSports::MarketBettingGroup
description: Format for BookieSports::MarketBettingGroup
type: object
properties:

 description:
```

(continues on next page)

(continued from previous page)

```
  $ref: "#/definitions/internationalized_name"

 asset:
  $ref: "#/definitions/asset"

 dynamic:
  description: Is this a dynamic BMG (like a NFL handicap or NBA Over-under BMG)?
  anyOf:
   - type: string
     enum:
      - ou  # Over under
      - hc  # handicap
   - type: boolean

 number_betting_markets:
  type: number
  description: Number of Betting Markets in this BMG
  items:
   type: string

 is_live:
  type: boolean
  description: WIll this BMG be turned Live at Event start? This is YES for all BMGs␣
↪at launch

 rules:
  type: string
  description: Human readable rules that the Grading Algorithm is a machine-readable␣
↪instantiation of

 bettingmarkets:
  type: array
  description: Betting markets to open
  items:
   type: object
   properties:
    description:
     $ref: "#/definitions/internationalized_name"

required:
 - description
 - asset
 - dynamic
 - number_betting_markets
 - is_live
 - rules
 - bettingmarkets
```

2.4 Naming Scheme

Some bookiesports files (in particular `name` and `description` fields) allow the use of *variables*. Those are dynamic and filled in by `bookie-sync`, automatically.

As an example, the file `MLB_ML_1.yaml` defines betting markets for a Moneyline market group. The betting markets carry the name of the event participants. We encode this in bookiesports using *variables*::

```
bettingmarkets:
     - description:
         en: '{teams.away}'
     - description:
         en: '{teams.home}'
```

2.4.1 Overview of variables

- `teams`:
  - `{teams.home}`: Home team
  - `{teams.away}`: Away team
- `result`:
  - `{teams.home}`: Points for home team
  - `{teams.away}`: Points for away team
  - `{teams.hometeam}`: Points for home team
  - `{teams.awayteam}`: Points for away team
  - `{teams.total}`: Total Points
- `handicaps`:
  - `{teams.home}`: Comparative (symmetric) Handicaps (e.g., +-*2*) for home team
  - `{teams.away}`: Comparative (symmetric) Handicaps (e.g., +-*2*) for away team
  - `{teams.home_score}`: Absolute handicap for home team (e.g., *2*)
  - `{teams.away_score}`: Absolute handicap for away team (e.g., *0*)
- `overunder`:
  - `{teams.value}`: The over-/under value

2.4.2 Internal Processing

The variable parsing is done in `bos-sync` (`substitutions.py`) and work through `decode_variables` and a few classes that deal with the variables. This allows us to have complex variable substitutions.

The variables all consist of a **module identifier** and the actual **member variable**::

```
{module.member}
```

All modules are listed in the `substitutions` variable in `decode_variables`::

```
substitutions = {
    "teams": Teams,
    "result": Result,
    "handicaps": Handicaps,
    "overunder": OverUnder,
}
```

The modules themselves (captical first letter) are defined in the same file and can be as easy as

```
class Result:
    """ Defines a few variables to be used in conjuctions with {result.X}
    """
    def __init__(self, **kwargs):
        result = kwargs.get("result", [0, 0]) or [0, 0]
        self.hometeam = result[0]
        self.awayteam = result[1]

        self.total = sum([float(x) for x in result])

        # aliases
        self.home = self.hometeam
        self.away = self.awayteam
```

and as complex as

```
class Teams:
    """ Defines a few variables to be used in conjuctions with {teams.X}
    """
    def __init__(self, **kwargs):
        teams = kwargs.get("teams", ["", ""]) or ["", ""]
        self.home = " ".join([
            x.capitalize() for x in teams[0].split(" ")])
        self.away = " ".join([
            x.capitalize() for x in teams[1].split(" ")])
```

CHAPTER 3

API

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index