Package 'betfairly'

February 15, 2013

Maintainer Vitalie Spinu <spinuvit@gmail.com>

License GPL (>= 2)

Title Access Betfair API from R

Type Package

LazyLoad yes

Author Vitalie Spinu <spinuvit@gmail.com>

Description An R interface to the Betfair API. The user can choose the type of output to be XML, S4 (representing native Betfair SOAP types), nested list or simplified R objects ready to be used in the statistical analysis.

Version 1.2

URL http://code.google.com/p/betfairly/

Date 2011-11-27

Depends R (>= 2.10), methods, RCurl, XML

Suggests XMLSchema

Collate 'betfair.R' 'classes.R' 'funcs.R'

Repository CRAN

Date/Publication 2011-11-29 06:57:30

NeedsCompilation no

R topics documented:

betfairly-package . . . 2
asBFDateTime . . . 4
betfair-class . . . 5
bfInitClasses . . . 5
bfSimpleOutput-class . . . 6
bfSimpleOutputDF-class . . . 7
bfSimpleOutputList-class . . . 8
BF_Acount_Management . . . 8
BF_Bet_History . . . 10
BF_Bet_Placement . . . 14
BF_Events . . . 18
BF_Markets . . . 20
BF_Prices . . . 22
BF_Session_Management . . . 23

Index **26**

`betfairly-package` *Access Betfair API from R*

Description

The `betfairly` package allows to access most of the Betfair API directly from R.

Details

For the list of all implemented functions and the details of the current development status please see todo.

If a particular functionality that you need is missing, please contact the author.

For examples of usage see here.

Table of most common functions:

If you want to:	Use:
Login	`bfLogin`
Request a list of available events	`getActiveEventTypes`
Request a list of Market for a specific eventType	`getAllMarkets`
Request details of a Market (excluding prices)	`getMarket`
Request prices for a Market	`getMarketPricesCompressed`
Place a bet	`placeBets`
Cancel a bet before it is matched	`cancelBets`
Retrieve a list of my Matched/Unmatched bets	`getMUBets`
Edit an Unmatched bet	`updateBets`
Retrieve the P&L for a market	`getMarketProfitAndLoss`
Place a Betfair SP bet	`placeBets`
Check if a market is in-play now	`getMarketPricesCompressed`

Check if a market is due to be turned in-play	getAllMarkets
Retrieve a list of Settled bets	getBetHistory
Retrieve your P&L for a market	getMarketProfitAndLoss

For a description of payed and free access types see http://bdp.betfair.com/index.php?option=com_content&task=view&id=36&Itemid=64.

Output of betfairly functions

All betfairly API functions can return four types of output, given by the option parameter which can be:

simple **(the default)** Simplified output represented by a bfSimpleOutput object containing slots bfType (original betfair class), errorCode ("OK" if succeed), and minorErrorCode (usually an empty string). See bfSimpleOutput-class for more information.

xml raw XML representation

list recursive list mirroring the structure of the node

S4 S4 object as described by the service SOAP protocol. Note what you will need XMLSchema package for the S4 conversion to work, as it defines some classes which are not provided with betfairly package. See bfInitClasses for further instructions.

You can change the default by setting the bfOutput option:

```
options(bfOutput = "XML")
```

Betfair exchange servers

Functions to betfair exchange services accept a server parameter, which can be either "GB" (the default) or "AU". You can set a different default with options(bfServer = "AU")

Curl Options

Each betfairly function accepts curlOpts parameter which is passed directly to curlPerform, see the documentation of that function for details.

Reporting Bugs

Before reporting bugs please see the relevant section in the official Betfair documentation and ensure it's not a betfair service issue. Known isssues are documented for each Betfair API action.

Disclaimer

The betfairly package is provided with absolutely no warranty. The documentation of the functional API is an adapted and abbreviated version of official Betfair documentation. Please refer to it for the complete reference.

Author(s)

Vitalie Spinu <spinuvit@gmail.com>

References

http://code.google.com/p/betfairly/, https://docs.developer.betfair.com/betfair/ Betfair API Quick Start

See Also

bfSimpleOutput-class

---

asBFDateTime *Convert R date-time objects into XMLSchema types as required by betfair interface.*

Description

Convert R date/date-time objects into valid XMLSchema representation as required by betfair interface.

Usage

```
asBFDateTime(x, tz="Z")
```

Arguments

x	Any R date/date-time object or character string which is recognized by as.POSIXlt.
tz	if supplied should be of the form (('+' | '-') hh ':' mm). For example +05:20 means 5 hours 20 minutes before the UTC. And +01:00 is equivalent to CEST.

Value

time string in appropriate format

Author(s)

Vitalie Spinu (<spinuvit@gmail.com>)

References

http://www.w3.org/TR/xmlschema-2/#dateTime

betfair-class *All betfairly S4 classes inherit from this class.*

Description

All betfairly S4 classes inherit from this class.

Details

If 'output' parameter is "S4", betfairly api functions return an S4 object. The structure of this object is described by the WSDL betfair service file.

For parsimony reasons the betfairly S4 classes are not installed with the package. You need to run `bfInitClasses()` to initialize the S4 interface.

Note what the package `XMLSchema` from www.omegahat.org is needed for this:

```
install.packages("XMLSchema")  ## binaries
```

or

```
install.packages("XMLSchema", repos = "http://www.omegahat.org/R", type = "source")
```

See Also

'betfairly-package' 'link{bfInitClasses}'

bfInitClasses *Initialize betfair S4 interface*

Description

All betfair functions are capable of producing a valid S4 object corresponding to betfair SOAP specification.
For parsimony reasons the betfair S4 classes are not installed with the package. You need to run `bfInitClasses()` to initialize them. Note that `XMLSchema` package is required for this initialization.

Usage

```
bfInitClasses(verbose=FALSE, where=.GlobalEnv)
```

Arguments

verbose	Print info message for each class
where	Environment in which to store the class definitions; defaults to global environment.

Value

'bfSimpleOutput' object, xml node or S4 object, as specified by the output parameter

Author(s)

Vitalie Spinu (<spinuvit@gmail.com>)

References

https://docs.developer.betfair.com/betfair/

See Also

'betfairly-package' 'bfSimpleOutput-class'

Examples

```
## Not run:
install.packages("XMLSchema", repos = "http://www.omegahat.org/R")  ## windows binaries?
install.packages("XMLSchema", repos = "http://www.omegahat.org/R", type = "source")  ## from source
bfInitClasses()

## End(Not run)
```

---

bfSimpleOutput-class *Virtual Class to represent the simplified output of betfairly functions...*

Description

Virtual Class to represent the simplified output of betfairly functions

Details

As described in 'betfairly-package' functions can return four types of output xml, S4, list or simplified output of class bfSimpleOutput.

There are two classes what inherit from bfSimpleOutput - bfSimpleOutputList and bfSimpleOutputDF. All betfairly functions return an object which extends one of these two classes. The names of the classes are always constructed by appending "Simple" or "SimpleDF" to the name of native Betfair class. For example the function getEvents returns an object of class GetEventsRespSimple meaning that it is a list inherited from bfSimpleOutputList and the native Betfair response type is GetEventsResp, so you can easily find the documentation in Betfair API reference guide. Function getAllMarkets return an object of class GetAllMarketsRespSimpleDF which means that it inherits from bfSimpleOutputDF and is a data.frame.

Slots

`bfType`: Name of Betfair SOAP type.

`errorCode`: Error code returned by Betfair api. You should check this first.

`minorErrorCode`: Age verification error

Prototype

`errorCode` = NA

`minorErrorCode` = NA

See Also

`bfSimpleOutputList-class`, `bfSimpleOutputDF-class`

Examples

```
getClass("bfSimpleOutput")
```

`bfSimpleOutputDF-class`

bfSimpleOutputDF is an S4 data...

Description

`bfSimpleOutputDF` is an S4 data.frame containing betfair tabular output.

Extends

`data.frame`, `bfSimpleOutput`

Author(s)

Vitalie Spinu

See Also

`betfairly-package` `bfSimpleOutput-class` `bfSimpleOutputList`

Examples

```
getClass("bfSimpleOutputDF")
```

`bfSimpleOutputList-class` *bfSimpleOutputList is an S4 list containing simple Betfair API output...*

Description

`bfSimpleOutputList` is an S4 list containing simple Betfair API output as familiar basic R types.

Details

Additional slots are usually data frames containing complex tabular data. For example an object `GetEventsRespSimple`, returned by function `getEvents`, contains two slots - `eventItems` and `marketItems`.

Methods

show `signature(object = "bfSimpleOutput")`: ...

Extends

`namedList`, `bfSimpleOutput`

Author(s)

Vitalie Spinu

See Also

`betfairly-package` `bfInitClasses`

Examples

```
getClass("bfSimpleOutputList")
```

`BF_Acount_Management` *Account management.*

Description

Various functions to access information about your account and wallets.

Usage

```
getAccountFunds(server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
getAccountStatement(startDate=Sys.Date() - 1, endDate=Sys.time(), startRecord=0,
    recordCount=100, itemsIncluded="ALL", locale,
    ignoreAutoTransfers=TRUE, server=getOption("bfServer"),
    output=getOption("bfOutput"), curlOpts=list())
getSubscriptionInfo(output=getOption("bfOutput"), curlOpts=list())
transferFunds(amount, sourceWalletId=1, targetWalletId=2,
    output=getOption("bfOutput"), curlOpts=list())
viewProfile(output=getOption("bfOutput"), curlOpts=list())
```

Arguments

server	"GB" (default) or "AU" - a Betfair exchange server to use. You can set the default with options(bfServer = "AU").
output	Indicates the form of the returned value. Can be "simple" (default), "xml", "list" or "S4". See betfairly-package.
curlOpts	RCurl options passed directly to curlPerform. You can also set the defaults with options(bfCurlOpts = list(opt1 = val1, opt2 = val2, ...)).
startDate	Return records on or after this date.
endDate	Return records on or before this date.
startRecord	The first record number to return (supports paging). Record numbering starts from 0. For example, to retrieve the third record and higher, set startRecord to 2.
recordCount	The maximum number of records to return.
itemsIncluded	Determines what type of statements items to return.
locale	Specify the language for the reply if you want a different language than the account default.
ignoreAutoTransfers	_
amount	_
sourceWalletId	The wallet that you are requesting the funds to be transferred from. There are two possible wallets: 1 = UK Sports Betting wallet 2 = Australian Sports Betting wallet
targetWalletId	The wallet that you are requesting the funds to be transferred from.

Details

getAccountFunds: Retrieve information about your local wallet on a particular exchange server. For an explanation of the concept of wallets, see "Using Region-specific Wallets for Placing Bets" on page 12 in Betfair API Developer Documentation.

getAccountStatement: Obtain information about transactions involving your local wallet on an exchange server.

getSubscriptionInfo: Return information on your API subscription.

transferFunds: Transfer funds between your UK and Australian account wallets. The concept of account wallets has been introduced in release 5.0 of the Betfair API. Instead of a single account holding all of a customer's funds for betting on sports events, there are now two "wallets" for each customer's account: one for betting on the UK exchange server and one for betting on the Australian exchange server.

viewProfile: Retrieve information about the user account, such as the registered address, e-mail address, phone numbers, etc.

Value

getAccountFunds: Object of class GetAccountFundsRespSimple with no extra slots.

getAccountStatement: Data.frame of class GetAccountStatementRespSimpleDF with no extra slots.

getSubscriptionInfo: Object of class getSubscriptionInfo with no extra slots.

transferFunds: Object of class TransferFundsRespSimple with no extra slots.

viewProfile: Object of class ViewProfileRespSimple with not extra slots.

Author(s)

Vitalie Spinu (<spinuvit@gmail.com>)

References

http://code.google.com/p/betfairly/, https://docs.developer.betfair.com/betfair/http://code.google.com/p/betfairly/, https://docs.developer.betfair.com/betfair/

See Also

betfairly-package bfSimpleOutput-classbetfairly-package bfSimpleOutput-class

---

BF_Bet_History *Access your bets*

Description

With getBetHistory, getMUBets and getMUBetsLite you access information about all your bets. With getBet, getBetLite and getBetMatchesLite you can access detailed information about your specific bets.

Usage

```
getBet(betId, server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
getBetLite(betId, server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
getBetMatchesLite(betId, server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
getMUBets(marketId, betIds, betStatus="MU", matchedSince, orderBy="BET_ID",
    sortOrder="ASC", recordCount=200, startRecord=0,
    excludeLastSecond=FALSE, server=getOption("bfServer"),
    output=getOption("bfOutput"), curlOpts=list())
getMUBetsLite(marketId, betIds, betStatus="MU", matchedSince, orderBy="BET_ID",
    sortOrder="ASC", recordCount=200, startRecord=0,
    excludeLastSecond=FALSE, server=getOption("bfServer"),
    output=getOption("bfOutput"), curlOpts=list())
getBetHistory(marketId=0, eventTypeIds, detailed=FALSE, sortBetsBy="NONE",
    betTypesIncluded="S", marketTypesIncluded="O",
    placedDateFrom=Sys.Date(), placedDateTo=Sys.time(),
    recordCount=100, startRecord=0, locale, timezone,
    server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
getMarketProfitAndLoss(marketID, includeSettledBets=FALSE, includeBspBets=TRUE,
    netOfCommission=FALSE, locale, server=getOption("bfServer"),
    output=getOption("bfOutput"), curlOpts=list())
```

Arguments

`betId`	The unique bet identifier
`server`	"GB" (default) or "AU" - a Betfair exchange server to use. You can set the default with `options(bfServer = "AU")`.
`output`	Indicates the form of the returned value. Can be "simple" (default), "xml", "list" or "S4". See `betfairly-package`.
`curlOpts`	RCurl options passed directly to `curlPerform`. You can also set the defaults with `options(bfCurlOpts = list(opt1 = val1, opt2 = val2, ...))`.
`marketId`	For **getMUBets** and **getMUBetsLite**: If marketId is present and non-zero, then bets placed on the specified market are returned and any bet ids specified in betIds are ignored. For **getBetHistory** Returns the records of your matched or unmatched bets for the specified market. If you use this parameter you must not specify `eventTypeId` array. Note that, if you specify a marketId, you must also specify either M or U as the value for the `betTypesIncluded` parameter.
`betIds`	Specifies the betId of each bet you want returned. The maximum number of bets you can include in the array is 200. If you include marketId in the request and marketId contains a non-zero value, then betIds is ignored. If you specify a betId, then you must specify MU for betStatus.

`betStatus` M, U or MU. The status of the bets to return (matched, unmatched, or both) - please see betfairly Simple Data Types . If you specify a betId, then you must specify MU.

`matchedSince` Specifies a date and time to start from for the list of returned bets. Any R date-time object or string recognized by as.POSIXlt. Use `asBFDateTime` to see how your time input is interpreted. If you use the matchedSince parameter and you have specified a betStatus of MU, the bets returned will ignore any limit you set (using recordCount) for the number of records to be returned. Specifying a betStatus of MU causes the API to return your unmatched bets along with the matched ones.

`orderBy` The order of returned results. Valid orders are BET_ID, PLACED_DATE, and MATCHED_DATE.

`sortOrder` ASC or DESC. Whether the results are in ascending or descending order

`recordCount` Maximum number of records to return. The maximum number allowed is 200.

`startRecord` The first record number to return (supports paging). Record numbering starts from 0. For example, to retrieve the third record and higher, set startRecord to 2.

`excludeLastSecond`
If true, the API excludes bets placed or matched that occurred less than one second before the GetMUBets call. Set this to true if you want to ensure that the response does not include bets that may have changed state between the time you sent the request and before the response was generated. If false, all bets are returned. Therefore, you may receive a response that indicates an unmatched bet that has actually been matched during the time taken for the API to respond.

`eventTypeIds` An array of event types to return. For matched and unmatched bets only, you can leave it unspecified and specify zero (the default) as the marketId to receive records of all your bets on the exchange.

`detailed` [logical] Whether to show details of all the matches on a single bet

`sortBetsBy` [ASC, DESC] How the bets are ordered.

`betTypesIncluded`
[C Cancelled, L Lapsed, M Matched, MU Matched and Unmatched, S Settled (default), U Unmatched, V Voided] Indicates the status of the bets to include in the response. If your `betHistory` request is for a specific market (in other words, if you have specified a marketId in your request), then you must specify either M or U as the value for betTypesIncluded. Otherwise you will receive an INVALID_BET_STATUS error. Only settled markets return cancelled, void, or lapsed bets.

`marketTypesIncluded`
[A Asian Handicap, L Line, O Odds (default), R Range] Indicates the types of market that you want your betting history returned for.

`placedDateFrom` Any R date/date-time object is accepted or any character string recognized by as.POSIXlt. Default to current day at 00:00.

`placedDateTo` Any R date/date-time object. Default to Sys.time().

`locale` Specify the language for the reply if you want a different language than the account default.

`timezone` Specify an alternative time-zone from the user account default.

`marketID` The market ID for which the profit and loss for the user is to be returned

`includeSettledBets`
logical If TRUE then the P&L calculation for each runner includes any profit and loss from any bets on runners that have already been settled. The default is FALSE, which matches the default on Betfair.com.

`includeBspBets` If TRUE, BSP bets are returned as part of the P&L

`netOfCommission`
If TRUE return P&L net of users current commission rate for this market including any special tariffs, default is FALSE.

Details

`getBet`: Retrieve information about a particular bet. Each request will retrieve all components of the desired bet.

You can retrieve Cancelled, Lapsed, and Voided bets from only settled markets and these bets are available for a maximum of 10 days from the date the market was settled.

`getBetLite`: This is the lite version of the GetBet service.

`getMUBets`: Retrieve information about all your matched and unmatched bets on a particular exchange server. You should be aware that voided bets are not returned by getMUBets. Your application should track the number of matched and unmatched bets against the number of bets returned by getMUBets in order to detect a voided bet.

`getMUBetsLite`: This is a lite version of the getMUBets service.

`getMarketProfitAndLoss`: Retrieve Profit and Loss information for the user account in a given market. The limitations for the service in the initial release are:

* Profit and loss for single and multi-winner odds markets is implemented however it won't calculate worstCaseIfWin nor futureIfWin.

* The calculation for AH markets will include worstCaseIfWin but not futureIfWin.

Value

`getBet`: A list of class `GetBetRespSimple` containing slot `matches` with info about matched portions of the bet.

`getBetLite`: A list of class `GetBetLiteRespSimple` with no additional slots. Contains a subset of information from data part of `getBet` response.

`getBetMatchesLite`: Data frame of class `GetBetMatchesLiteRespSimple` containing subset of information from `@matches` slot in `getBet` response.

`getMUBets`: Object of class `xxx` containing slot

`getBetHistory`: Object of class `GetBetHistoryRespSimple` containing slots `betHistoryItems` - a data frame with one bet per row and `matches` - a data frame with all the matches if the `details` parameter was set to TRUE.

`getMarketProfitAndLoss`: Object of class `GetMarketProfitAndLossRespSimple` containing slot `annotations` which is a data frame with P&L data.

Author(s)

Vitalie Spinu (<spinuvit@gmail.com>)

References

http://code.google.com/p/betfairly/, https://docs.developer.betfair.com/betfair/http://code.google.com/p/betfairly/, https://docs.developer.betfair.com/betfair/http://code.google.com/p/betfairly/, https://docs.developer.betfair.com/betfair/

See Also

betfairly-package bfSimpleOutput-classbetfairly-package bfSimpleOutput-classbetfairly-package bfSimpleOutput-class

---

BF_Bet_Placement *Functions to place, update and cancel bets.*

Description

Place, update and cancel multiple bets at a time.

For placeBets and updateBets, there are two equivalent ways of supplying the bet info. First, by passing a list of corresponding objects (bfBet and bfBetUpdate) as bets argument. Second, a vectorized (mapplyish) way, by supplying vectors to corresponding arguments. Vectorized arguments are recycled to the same length if needed. These arguments are all in plural and are not documented below.

Usage

```
bfBet(marketId, selectionId, price, size, betType="B", bspLiability=0,
    betCategoryType="E", betPersistenceType="NONE",
    server=getOption("bfServer"))
placeBets(bets=list(), marketIds, selectionIds, prices, sizes, betTypes="B",
    bspLiabilities=0, betCategoryTypes="E", betPersistenceTypes="NONE",
    server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
cancelBets(bets, server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
cancelBetsByMarket(markets, server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
bfBetUpdate(betId, newPrice, oldPrice, newSize, oldSize, newBetPersistenceType,
    oldBetPersistenceType)
updateBets(bets=list(), betIds, newPrices, oldPrices, newSizes, oldSizes,
    newBetPersistenceTypes, oldBetPersistenceTypes,
    server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
```

Arguments

marketId	Integer specifying the market ID.
selectionId	ID of the desired runner or selection within the market
price	numeric The price (odds) you want to set for the bet. Valid values are 1.01 to 1000. For a BSP Limit on Close bet, specify the desired price limit. For a Back bet, the minimum price you want. If the Starting Price is lower than this amount, then your bet is not matched. For a Lay bet, the maximum acceptable price. If the Starting Price is higher than this amount, then your bet is not matched. If the specified limit is equal to the starting price, then it may be matched, partially matched, or may not be matched at all, depending on how much is needed to balance all bets against each other (MOC, LOC and normal exchange bets).
size	numeric The stake (amount) for an exchange bet. The minimum amount for a back bet is 2 (or equivalent). If the betPersistenceType is set to SP, then the minimum amount for a lay bet is 10 (or equivalent), otherwise, the minumum lay bet amount is 2 (or equivalent).
betType	'B' - back, 'L'- lay. See details.
bspLiability	numeric This is the maximum amount of money you want to risk for a BSP bet. The minimum amount for a back bet is 2 (or equivalent). The minimum amount for a lay bet is 10 (or equivalent) For a back bet, this is equivalent to the stake on a normal exchange bet. For a lay bet, this is the equivalent to the liability on a normal exchange bet. If after the market is reconciled, the actual stake is calculated once the price is known.
server	"GB" (default) or "AU" - a Betfair exchange server to use. You can set the default with `options(bfServer = "AU")`.
betCategoryType	E, M or L. 'E' - Exchange bet, 'M' - Market on Close SP bet, 'L' - Limit on Close SP bet. If you specify Limit on Close, specify the desired limit using the price argument. See details.
betPersistenceType	NONE, IP or SP. Specify what happens to an unmatched (or partially unmatched) exchange bet when the market turns in-play. If betCategoryType is an SP bet, betPersistenceType must be set to NONE. See details.
bets	For `placeBets` an `bfBet` object or a list (of max 60) such objects. For `cancelBets` a vector of bet ids to be canceled (max 40). For `updateBets` an `bfBetUpdate` object or a list (of max 15) such objects.
marketIds	Vector of integers specifying the market IDs.
selectionIds	_
prices	_
sizes	_
betTypes	_
bspLiabilities	_
betCategoryTypes	_

betPersistenceTypes
: _

output
: Indicates the form of the returned value. Can be "simple" (default), "xml", "list" or "S4". See betfairly-package.

curlOpts
: RCurl options passed directly to curlPerform. You can also set the defaults with options(bfCurlOpts = list(opt1 = val1, opt2 = val2, ...)).

markets
: Vector of market IDs.

betId
: The unique identifier for the bet.

newPrice
: New odds desired on the bet For BSP Limit on Close bets, newPrice should be set to the new limit desired. For BSP Limit on Close back bets, you can only change the limit downward. For BSP Limit on Close lay bets, you can only change the limit upward.

oldPrice
: For an exchange bet, original odds on the bet.

newSize
: New stake desired on the bet

oldSize
: For an exchange bet, original stake on the bet

newBetPersistenceType
: New persistence type on an exchange bet. Only valid before the market turns in-play.

oldBetPersistenceType
: Original persistence type on an exchange bet. Only valid before the market turns in-play.

betIds
: _

newPrices
: _

oldPrices
: _

newSizes
: _

oldSizes
: _

newBetPersistenceTypes
: _

oldBetPersistenceTypes
: _

Details

bfBet: Constructor of bfBet object. You supply a list of these objects as bets argument to placeBets.

##' The required fields in bets are dependent on the category of bet. The following table shows the required fields for each bet category.

Table 1. Valid Bet Category request field combinations

Bet Category	Price	Size	BspLiability	BetPersistenceType
Exchange	Yes	Yes	No	Yes
Market on Close	No	No	Yes	No
Limit on Close	Yes	No	Yes	No

`placeBets`: Place multiple (1 to 60) bets on a single Market. There is an instance of PlaceBetsResp returned in the output for each instance of PlaceBets in the input. The success or failure of the individual bet placement operation is indicated by the Success Boolean.

Bet Types You can specify, for each bet, if you want to place a Back bet or a Lay bet.

* B - Back bets win when the selection is settled as the winner in the market. * L - Lay bets win when the selection is settled as a looser in the market.

For more information on Bet types, see the Betfair website help.

Bet Categories You can specify, for each bet, whether the bet is a regular exchange bet, or a Betfair Market on Close (or Starting Price) bet (with or without a price limit).

* E - Exchange bets are placed on the market and are matched against bets at the specified or better price. Exchange bets are matched on a first in, first matched basis.

* M - Market on Close (MOC) bets remain unmatched until the market is reconciled. They are matched and settled at a price that is representative of the market at the point the market is turned in-play. The market is reconciled to find a starting price and MOC bets are settled at whatever starting price is returned. MOC bets are always matched and settled, unless a starting price is not available for the selection. Market on Close bets can only be placed before the starting price is determined.

* L - Limit on Close (LOC) bets are matched if, and only if, the returned starting price is better than a specified price. In the case of back bets, LOC bets are matched if the calculated starting price is greater than the specified price. In the case of lay bets, LOC bets are matched if the starting price is less than the specified price. If the specified limit is equal to the starting price, then it may be matched, partially matched, or may not be matched at all, depending on how much is needed to balance all bets against each other (MOC, LOC and normal exchange bets)

Bet Persistence You can specify what happens to an Exchange bet that is unmatched when the market is reconciled and the starting price is calculated.

* NONE - The unmatched bet is cancelled when the market is reconciled and turned in-play.

* IP - The unmatched bet stays as an unmatched bet when the market is turn in-play.

* SP - The unmatched bet becomes a Market on Close bet and is matched at the starting price.

`cancelBets`: Cancel multiple unmatched (1 to 40) bets placed on a single Market. The success or failure of the individual bet cancellation operation will be indicated by the Success Boolean. If a portion of the original bet is already matched, cancelBets cancels the unmatched portion of the bet.

`cancelBetsByMarket`: [payed] Cancel all unmatched bets (or unmatched portions of bets) placed on one or more Markets. You might use this service to quickly close out a position on a market.

`bfBetUpdate`: Constructor of `bfBetUpdate` object. You supply a list of these objects as `bets` argument to `cancelBets`.

`updateBets`: Edit multiple (1 to 15) bets on a single Market. The success or failure of the individual bet editing operation is indicated by the Success Boolean.

If newPrice and newSize are both specified the newSize value is ignored. For example, an original bet is placed for 100 with odds of 1.5: UpdateBets is called with newSize = 200, newPrice = 2. The original bet is cancelled and a new bet is place for 100 with odds of 2.

Value

bfBet: String of class bfBet.

placeBets: Data frame with info on the success of placed bets, one bet per row.

cancelBets: Data frame with info on canceled bets, one bet per row.

cancelBetsByMarket: Object of class xxx containing slot

bfBetUpdate: String of class bfBetUpdate.

updateBets: Data frame with info on the success of bet updates, one bet per row.

Author(s)

Vitalie Spinu (<spinuvit@gmail.com>)

References

http://code.google.com/p/betfairly/, https://docs.developer.betfair.com/betfair/

See Also

betfairly-package bfSimpleOutput-class

BF_Events *Betfair Events*

Description

Functions to retrieve betfair events (Games, sports, politics etc)

Usage

```
getAllEventTypes(locale, output=getOption("bfOutput"), curlOpts=list())
getActiveEventTypes(locale, output=getOption("bfOutput"), curlOpts=list())
getEvents(eventParentId=1, locale, output=getOption("bfOutput"), curlOpts=list())
```

Arguments

locale	Specify the language for the reply if you want a different language than the account default.
output	Indicates the form of the returned value. Can be "simple" (default), "xml", "list" or "S4". See betfairly-package.
curlOpts	RCurl options passed directly to curlPerform. You can also set the defaults with options(bfCurlOpts = list(opt1 = val1, opt2 = val2, ...)).
eventParentId	integer This is either an Id value for a single item (in an array of eventTypeItems returned by GetAllEventTypes or GetActiveEventTypes), or it is an eventId for a single eventItem (in an array of eventItems returned by an earlier GetEvents request).

Details

`getAllEventTypes`: Allows the customer to retrieve lists of all categories of sports (Games, Event Types) that have at least one market associated with them, regardless of whether that market is now closed for betting. This means that, for example, the service would always return the event types Soccer and Horse Racing and would also return Olympics 2004 or EURO 2004 for a certain period after the markets for those events had closed; it would also return Olympics 2004 or EURO 2004 for a certain period before the markets for those events had opened. The service returns information on future events to allow API programmers to see the range of events that will be available to bet on in the near future.

`getActiveEventTypes`: Allows the customer to retrieve lists of all categories of sporting events (Games, Event Types) that are available to bet on: in other words, all those that have at least one currently active or suspended market associated with them. This means, therefore, that the service would, for example, always return the event types Soccer and Horse Racing but would not return Olympics 2004 or EURO 2004 after those events had finished.

`getEvents`: Allows you to navigate through the events hierarchy until you reach details of the betting market for an event that you are interested in.

From API 5.0 onwards, the GetEvents service returns details of line and range markets, where these markets are available for an event. Requests for the GetEvents service take as input a parameter called eventParentID. The value of this parameter is either: the (integer) Id value from one item in an array of eventTypeItems that has been returned by the GetAllEventTypes or GetActiveEventTypes services; or an (integer) eventId value from one item in an array of eventItems that has been returned by an earlier GetEvents request. Use the GetEvents service repeatedly, specifying a different value for eventParentId in each request, until there are no further events to request (this means you have reached the leafnode of the branch of the events tree you have been navigating). To retrieve full details of a betting market whose details have been returned by the GetEvents service, you need to send a GetMarket request to the exchange server indicated by the market's exchangeId parameter (see CROSS REFERENCE TEXT NEEDS RESOLVING). This GetMarket request must also specify the marketId for the market you are requesting. Both the exchangeId and the marketId are returned by GetEvents. For information about GetMarket, see Chapter 24 .

Value

`getAllEventTypes`: A data frame with columns id nextMarketId and exchangeId; an xml node or S4 object, as specified by the `output` parameter

`getActiveEventTypes`: A data frame with columns id, nextMarketId and exchangeId; an xml node or an S4 object, as specified by the `output` parameter

`getEvents`: Object of class `GetEventsRespSimple` which inherits from `bfSimpleOutput` class. With slots `eventItems` and `marketItems` which are data.frames.

Note

The GetActiveEventTypes service is a global service, and it returns information about the events available on both the UK and the Australian exchange servers.

Author(s)

Vitalie Spinu (<spinuvit@gmail.com>)

References

http://code.google.com/p/betfairly/, https://docs.developer.betfair.com/betfair/

See Also

betfairly-package bfSimpleOutput-class getActiveEventTypes

BF_Markets *Betfair markets*

Description

Functions to retrieve information about Betfair markets.

Usage

```
getAllMarkets(eventTypeIds, countries, fromDate, toDate, locale,
    server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
getMarket(marketId, includeCouponLinks=FALSE, locale="en",
    server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
getMarketInfo(marketId, server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
getMarketTradedVolume(marketId, selectionId, asianLineId, currencyCode,
    server=getOption("bfServer"), output=getOption("bfOutput"),
    curlOpts=list())
getMarketTradedVolumeCompressed(marketId, currencyCode, server=getOption("bfServer"),
    output=getOption("bfOutput"), curlOpts=list())
```

Arguments

eventTypeIds	A vector with the events ids to return. If not specified, markets from all event types are returned.
countries	The countries where the event is taking place as an array of ISO3 country codes. If not specified, markets from all countries (or international markets) for the specified exchange are returned.
fromDate	Any R date-time object or string recognized by as.POSIXlt. Use asBFDateTime to see how your time input is interpreted. If this is set, the response contains only markets where the market time is not before the specified date.
toDate	Any R date-time object or string recognized by as.POSIXlt. If this is set, the response contains only markets where the market time is not after the specified date. No limit if not specified.
locale	Specify the language for the reply if you want a different language than the account default.

server	"GB" (default) or "AU" - a Betfair exchange server to use. You can set the default with options(bfServer = "AU").
output	Indicates the form of the returned value. Can be "simple" (default), "xml", "list" or "S4". See betfairly-package.
curlOpts	RCurl options passed directly to curlPerform. You can also set the defaults with options(bfCurlOpts = list(opt1 = val1, opt2 = val2, ...)).
marketId	Integer specifying the market ID.
includeCouponLinks	If you set this parameter to true, the service response contains a list of any coupons that include the market you have requested. If you set the parameter to FALSE (the default), no coupon data is returned.
selectionId	The desired runner id.
asianLineId	Mandatory if the market specified by Market ID is an Asian Market, otherwise optional
currencyCode	Three letter ISO 4217 code.

Details

getAllMarkets: Retrieve information about all of the markets that are currently active or suspended on the given exchange. You can use this service to quickly analyse the available markets on the exchange, or use the response to build a local copy of the Betfair.com navigation menu. You can limit the response to a particular time period, country where the event is taking place, and event type. Otherwise, the service returns all active and suspended markets.

getMarket: The API GetMarket service allows the customer to input a Market ID and retrieve all static market data for the market requested. To get a Market ID for the betting market associated with an event you are interested in, use the GetEvents command.

getMarketInfo: The API GetMarketInfo service allows you to input a Market ID and retrieve market data for the market requested. To get a Market ID for the betting market associated with an event you are interested in, use the GetEvents command. This is a lite service to compliment the GetMarket service.

getMarketTradedVolume: Obtain all the current odds and matched amounts on a single runner in a particular event.

getMarketTradedVolumeCompressed: Obtain the current price (odds) and matched amounts at each price on all of the runners in a particular market.

Value

getAllMarkets: A data.frame containing one market per row and a character string if output = "S4".

getMarket: Object of class GetMarketRespSimple which inherits from bfSimpleOutput class. Additional slot runners contains a data frame of event participants.

Object of native betfair class GetMarketResp if output = "S4".

getMarketInfo: Object of class GetMarketInfoRespSimple which inherits from bfSimpleOutput class and has no extra slots.

If output = "S4", object of native betfair class GetMarketInfoResp.

getMarketTradedVolume: Object of class GetMarketTradedVolumeRespSimple with a slot priceItems containing a data frame of total match volumes for each odd.

getMarketTradedVolumeCompressed: Object of class GetMarketTradedVolumeCompressedRespSimple with two additional slots runners and volumes.

Author(s)

Vitalie Spinu (<spinuvit@gmail.com>)

References

http://code.google.com/p/betfairly/, https://docs.developer.betfair.com/betfair/

See Also

betfairly-package bfSimpleOutput-class

BF_Prices *Prices on betfair markets.*

Description

Functions to retrieve prices on Betfair markets.

Usage

```
getCompleteMarketPricesCompressed(marketId, currencyCode="EUR", server=getOption("bfServer"),
    output=getOption("bfOutput"), curlOpts=list())
getMarketPrices(marketId, currencyCode, server=getOption("bfServer"),
    output=getOption("bfOutput"), curlOpts=list())
getMarketPricesCompressed(marketId, currencyCode, server=getOption("bfServer"),
    output=getOption("bfOutput"), curlOpts=list())
```

Arguments

marketId	Integer specifying the market ID.
currencyCode	Three letter ISO 4217 code.
server	"GB" (default) or "AU" - a Betfair exchange server to use. You can set the default with options(bfServer = "AU").
output	Indicates the form of the returned value. Can be "simple" (default), "xml", "list" or "S4". See betfairly-package.
curlOpts	RCurl options passed directly to curlPerform. You can also set the defaults with options(bfCurlOpts = list(opt1 = val1, opt2 = val2, ...)).

Details

getCompleteMarketPricesCompressed: Retrieve all back and lay stakes for each price on the exchange for a given Market ID in a compressed format. The information returned is similar to the GetDetailAvailableMarketDepth, except it returns the data for an entire market, rather than just one selection.

getMarketPrices: Retrieve dynamic market data for a given Market ID.

getMarketPricesCompressed: Retrieve dynamic market data for a given Market ID in a compressed format. This service returns the same information as the Get Market Prices service but returns it in a ~ (tilde) delimited String.

Value

getCompleteMarketPricesCompressed: A list of class GetCompleteMarketPricesCompressedRespSimple with three additional slots containing data.frames removedRunners, runners and prices. Use merge for joining these by the common field runners.

getMarketPrices: Object of class "GetMarketPricesRespSimple" with a slot runnerPrices containing a data frame of back and lay prices for each runner. This function returns the same information as getMarketPricesCompressed but in a merged, long format.

getMarketPricesCompressed: Object of class GetMarketPricesCompressedRespSimple containing slots runners and prices.

Author(s)

Vitalie Spinu (<spinuvit@gmail.com>)

References

http://code.google.com/p/betfairly/, https://docs.developer.betfair.com/betfair/

See Also

betfairly-package bfSimpleOutput-class getActiveEventTypes

BF_Session_Management *Session Management.*

Description

login enables customers to log in to the API service and initiates a secure session for the user. Users can have multiple sessions 'alive' at any point in time.

logout allows you to explicitly end your session.

keepAlive can be used to stop a session timing out. Every call to the Betfair API returns a token, in the sessionToken field, that identifies a login session. Every time your application calls the

Betfair API and is returned a sessionToken, the session timeout is reset to approximately 20 minutes. After the timeout has passed, the session is expired and you need to login again.
If you want to keep your login session active, but your application has not made any Betfair API calls that would generate a new sessionToken and reset the session timeout, you can call keepAlive to obtain a new sessionToken and reset the session timeout.

`bfSessionTocken` returns the current session token, if any, NULL otherwise.

Usage

```
bfLogin(username, password, productId=82, ipAddress="0", locationId=0, vendorSoftwareId=0, curlOpts
bfLogout(curlOpts = list())
keepAlive(curlOpts = list())
bfSessionToken()
bfSessionHandler()
```

Arguments

`username`	The username with which to login to the API for a new session.
`password`	The password with which to login to the API for a new session.
`productId`	The API product ID with which to login to the API for a new session. If you want to use the Free Access API, use 82. If you are a paying API subscriber, use the Id provided when you signed up.
`ipAddress`	For applications that proxy the user's connection, the IP address of the user's computer. Betfair may inform you in the future if you need to provide this field, otherwise set this field to 0 (the default).
`locationId`	The location ID with which to login for a new session.
`vendorSoftwareId`	The vendor software ID with which to login to the API for a new session. This is only relevant for software vendors and is provided when software vendors sign up.
`curlOpts`	RCurl options passed directly to `curlPerform`. You can also set the defaults with `options(bfCurlOpts = list(opt1 = val1, opt2 = val2, ...))`.

Details

`bfSessionHandler` (not implemented yet) creates a session handler used to access multiple sessions. A betfair functions can be accessed through `obj$foo(...)`, where `obj` is the session handler object returned by `bfSessionHandler`. All the functions in the handler share common parent environment where the `.sessionToken` is stored.

Every betfair API request/response must contains a session token. All functions in betfairly-package store the session token in .GlobalEnv in .sessionToken variable by <<- assignment. To manage a single account this is appropriate. To manipulate several different sessions at the same time create a handler for each session with `bfSessionHandler`.

Value

invisibly a sessionToken string

Author(s)

Vitalie Spinu (<spinuvit@gmail.com>)

References

http://code.google.com/p/betfairly/, https://docs.developer.betfair.com/betfair/

See Also

betfairly-package bfSimpleOutput-class

Index

∗Topic **api**
betfairly-package, 2
∗Topic **betfair**
betfairly-package, 2
∗Topic **class**
betfair-class, 5
bfSimpleOutput-class, 6
bfSimpleOutputDF-class, 7
bfSimpleOutputList-class, 8
∗Topic **package**
betfairly-package, 2
>BF_Acount_Management *(BF_Acount_Management)*, 8
>BF_Bet_Placement *(BF_Bet_Placement)*, 14
>BF_Events *(BF_Events)*, 18
>BF_Markets *(BF_Markets)*, 20
>BF_Prices *(BF_Prices)*, 22
>BF_Session_Management *(BF_Session_Management)*, 23

asBFDateTime, 4

betfair *(betfair-class)*, 5
betfair-class, 5
betfairly-class *(betfair-class)*, 5
betfairly-package, 2
BF_Acount_Management, 8
BF_Bet_History, 10
BF_Bet_Placement, 14
BF_Events, 18
BF_Markets, 20
BF_Prices, 22
BF_Session_Management, 23
bfBet *(BF_Bet_Placement)*, 14
bfBetUpdate *(BF_Bet_Placement)*, 14
bfInitClasses, *3*, 5, *8*
bfLogin, *2*
bfLogin *(BF_Session_Management)*, 23
bfLogout *(BF_Session_Management)*, 23
bfSessionHandler *(BF_Session_Management)*, 23
bfSessionToken *(BF_Session_Management)*, 23
bfSimpleOutput, *3*, *6–8*, *19*, *21*
bfSimpleOutput *(bfSimpleOutput-class)*, 6
bfSimpleOutput-class, 6
bfSimpleOutputDF, *6*
bfSimpleOutputDF *(bfSimpleOutputDF-class)*, 7
bfSimpleOutputDF-class, 7
bfSimpleOutputList, *6*, *7*
bfSimpleOutputList *(bfSimpleOutputList-class)*, 8
bfSimpleOutputList-class, 8

cancelBets, *2*
cancelBets *(BF_Bet_Placement)*, 14
cancelBetsByMarket *(BF_Bet_Placement)*, 14
curlPerform, *3*, *9*, *11*, *16*, *18*, *21*, *22*, *24*

data.frame, *7*

getAccountFunds *(BF_Acount_Management)*, 8
getAccountStatement *(BF_Acount_Management)*, 8
getActiveEventTypes, *2*, *20*, *23*
getActiveEventTypes *(BF_Events)*, 18
getAllEventTypes *(BF_Events)*, 18
getAllMarkets, *2*, *3*, *6*
getAllMarkets *(BF_Markets)*, 20
getBet *(BF_Bet_History)*, 10
getBetHistory, *3*
getBetHistory *(BF_Bet_History)*, 10
getBetLite *(BF_Bet_History)*, 10
getBetMatchesLite *(BF_Bet_History)*, 10
getCompleteMarketPricesCompressed *(BF_Prices)*, 22

getEvents, *6*, *8*
getEvents *(*BF_Events*)*, 18
getMarket, *2*
getMarket *(*BF_Markets*)*, 20
getMarketInfo *(*BF_Markets*)*, 20
getMarketPrices *(*BF_Prices*)*, 22
getMarketPricesCompressed, *2*
getMarketPricesCompressed *(*BF_Prices*)*, 22
getMarketProfitAndLoss, *2, 3*
getMarketProfitAndLoss *(*BF_Bet_History*)*, 10
getMarketTradedVolume *(*BF_Markets*)*, 20
getMarketTradedVolumeCompressed *(*BF_Markets*)*, 20
getMUBets, *2*
getMUBets *(*BF_Bet_History*)*, 10
getMUBetsLite *(*BF_Bet_History*)*, 10
getSubscriptionInfo *(*BF_Acount_Management*)*, 8

keepAlive *(*BF_Session_Management*)*, 23

merge, *23*

namedList, *8*

placeBets, *2*
placeBets *(*BF_Bet_Placement*)*, 14

transferFunds *(*BF_Acount_Management*)*, 8

updateBets, *2*
updateBets *(*BF_Bet_Placement*)*, 14

viewProfile *(*BF_Acount_Management*)*, 8