pandas$_{m}arket_{c}alendars$

Sep 06, 2023

Contents

Market calendars to use with pandas for trading applications.

CHAPTER 1

Documentation

http://pandas-market-calendars.readthedocs.io/en/latest/

CHAPTER 2

Overview

The Pandas package is widely used in finance and specifically for time series analysis. It includes excellent functionality for generating sequences of dates and capabilities for custom holiday calendars, but as an explicit design choice it does not include the actual holiday calendars for specific exchanges or OTC markets.

The pandas_market_calendars package looks to fill that role with the holiday, late open and early close calendars for specific exchanges and OTC conventions. pandas_market_calendars also adds several functions to manipulate the market calendars and includes a date_range function to create a pandas DatetimeIndex including only the datetimes when the markets are open. Additionally the package contains product specific calendars for future exchanges which have different market open, closes, breaks and holidays based on product type.

This package provides access to over 50+ unique exchange calendars for global equity and futures markets.

This package is a fork of the Zipline package from Quantopian and extracts just the relevant parts. All credit for their excellent work to Quantopian.

2.1 Major Releases

As of v1.0 this package only works with Python3. This is consistent with Pandas dropping support for Python2.

As of v1.4 this package now has the concept of a break during the trading day. For example this can accommodate Asian markets that have a lunch break, or futures markets that are open 24 hours with a break in the day for trade processing.

As of v2.0 this package provides a mirror of all the calendars from the exchange_calendars package, which itself is the now maintained fork of the original trading_calendars package. This adds over 50 calendars.

As of v3.0, the function date_range() is more complete and consistent, for more discussion on the topic refer to PR #142 and Issue #138.

As of v4.0, this package provides the framework to add interruptions to calendars. These can also be added to a schedule and viewed using the new interruptions_df property. A full list of changes can be found in PR #210.

2.2 Source location

Hosted on GitHub: https://github.com/rsheftel/pandas_market_calendars

2.3 Installation

```
pip install pandas_market_calendars
```

Arch Linux package available here: https://aur.archlinux.org/packages/python-pandas_market_calendars/

CHAPTER 3

Calendars

The list of available calendars

CHAPTER 4

Quick Start

```
import pandas_market_calendars as mcal

# Create a calendar
nyse = mcal.get_calendar('NYSE')

# Show available calendars
print(mcal.get_calendar_names())
```

```
early = nyse.schedule(start_date='2012-07-01', end_date='2012-07-10')
early
```

```
                 market_open              market_close
=========== ========================= =========================
 2012-07-02 2012-07-02 13:30:00+00:00 2012-07-02 20:00:00+00:00
 2012-07-03 2012-07-03 13:30:00+00:00 2012-07-03 17:00:00+00:00
 2012-07-05 2012-07-05 13:30:00+00:00 2012-07-05 20:00:00+00:00
 2012-07-06 2012-07-06 13:30:00+00:00 2012-07-06 20:00:00+00:00
 2012-07-09 2012-07-09 13:30:00+00:00 2012-07-09 20:00:00+00:00
 2012-07-10 2012-07-10 13:30:00+00:00 2012-07-10 20:00:00+00:00
```

```
mcal.date_range(early, frequency='1D')
```

```
DatetimeIndex(['2012-07-02 20:00:00+00:00', '2012-07-03 17:00:00+00:00',
               '2012-07-05 20:00:00+00:00', '2012-07-06 20:00:00+00:00',
               '2012-07-09 20:00:00+00:00', '2012-07-10 20:00:00+00:00'],
              dtype='datetime64[ns, UTC]', freq=None)
```

```
mcal.date_range(early, frequency='1H')
```

```
DatetimeIndex(['2012-07-02 14:30:00+00:00', '2012-07-02 15:30:00+00:00',
               '2012-07-02 16:30:00+00:00', '2012-07-02 17:30:00+00:00',
```

(continues on next page)

(continued from previous page)

```
                '2012-07-02 18:30:00+00:00', '2012-07-02 19:30:00+00:00',
                '2012-07-02 20:00:00+00:00', '2012-07-03 14:30:00+00:00',
                '2012-07-03 15:30:00+00:00', '2012-07-03 16:30:00+00:00',
                '2012-07-03 17:00:00+00:00', '2012-07-05 14:30:00+00:00',
                '2012-07-05 15:30:00+00:00', '2012-07-05 16:30:00+00:00',
                '2012-07-05 17:30:00+00:00', '2012-07-05 18:30:00+00:00',
                '2012-07-05 19:30:00+00:00', '2012-07-05 20:00:00+00:00',
                '2012-07-06 14:30:00+00:00', '2012-07-06 15:30:00+00:00',
                '2012-07-06 16:30:00+00:00', '2012-07-06 17:30:00+00:00',
                '2012-07-06 18:30:00+00:00', '2012-07-06 19:30:00+00:00',
                '2012-07-06 20:00:00+00:00', '2012-07-09 14:30:00+00:00',
                '2012-07-09 15:30:00+00:00', '2012-07-09 16:30:00+00:00',
                '2012-07-09 17:30:00+00:00', '2012-07-09 18:30:00+00:00',
                '2012-07-09 19:30:00+00:00', '2012-07-09 20:00:00+00:00',
                '2012-07-10 14:30:00+00:00', '2012-07-10 15:30:00+00:00',
                '2012-07-10 16:30:00+00:00', '2012-07-10 17:30:00+00:00',
                '2012-07-10 18:30:00+00:00', '2012-07-10 19:30:00+00:00',
                '2012-07-10 20:00:00+00:00'],
               dtype='datetime64[ns, UTC]', freq=None)
```

CHAPTER 5

Contributing

All improvements and additional (and corrections) in the form of pull requests are welcome. This package will grow in value and correctness the more eyes are on it.

To add new functionality please include tests which are in standard pytest format.

Use pytest to run the test suite.

For complete information on contributing see CONTRIBUTING.md

CHAPTER 6

Future

This package is open sourced under the MIT license. Everyone is welcome to add more exchanges or OTC markets, confirm or correct the existing calendars, and generally do whatever they desire with this code.

CHAPTER 7

Sponsor

TradingHours.com provides the most accurate and comprehensive coverage of market holidays and trading hours data available. They cover over 900 markets around the world. Their data is continually monitored for changes and updated daily. Learn more

CHAPTER 8

Updates

8.1 Change Log

8.1.1 Updates

4.3.1 (09/06/2023)

- Fixed broken build PR #292

4.3.0 (09/05/2023)

- Fixed for pandas 2.0 so all tests pass PR #282
- Move exchange_calendar*.py files to pandas_market_calendar/exchange_calendars/ PR #284
- Move holidays_*.py to pandas_market_calendar/holidays/ PR #284
- Major cleanup including unused imports PR #284

4.2.1 (08/21/2023)

- Fix the pyproject.toml to properly generate sdist PR #267
- Remove .travis.yml file as Travis-CI is no longer used
- Merge .coveragerc into pyproject.toml

4.2.0 (08/20/2023)

- CBOE GoodFriday special close is broken, reverted back to standard GoodFriday logic PR #265
- Fixed BSE Holiday PR #248 Issue #245

- Updated TASE Holidays 2022-2025 PR #263
- King Charles III's Coronation Day holiday to LSE calendar PR #255
- Added NYSE tests for 2024 and 2025 PR #259
- Deleted setup.cfg that was only used for flake8. Will move to Black in a future release
- Moved all setuptools build workflows to pyproject.toml and deleted setup.py

4.1.4 (02/04/2023)

- Updated TASE Holidays 2022-2025

4.1.3 (12/26/2022)

- Added Chinese 2023 holidays

4.1.2 (12/08/2022)

- Added 2023 holidays to BSE calendar

4.1.1 (10/31/2022)

- Fix for bug in NYSEExchangeCalendar.valid_days

4.1.0 (10/08/2022)

- Added UK and Australia holidays for Queen Elizabeth II's State Funeral

4.0.3 (10/08/2022)

- Enabled tests that failed before PR #215

4.0.2 (10/08/2022)

- Implemented new release management

4.0.1 (09/03/22)

- Fix duplicates bug in special_dates
- Fix tz=None bug in NYSEExchangeCalendar.valid_days

4.0 (08/02/22)

- Added interruptions support
- Updated MarketCalendar.open_at_time to respect interruptions
- Special times can be set with offsets
- MarketCalendar.days_at_time returns a pandas.Series
- calendar_utils.date_range supports schedules of any timezone

3.5 (06/25/22)

- Updated BMF
- New CME calendar setup
- New CME calendars for equities, fixed income, ags, energies, metals, and FX

3.4 (03/05/22)

- Update to work with pandas 1.4.0
- Fix boxing day for Australia
- Add SIFMA US, UK and JP calendars
- Add IEX calendar
- Add NSE calendar

3.3 (01/30/22)

- PR #166 to solve the issue raised in #164
- Add Juneteenth to NYSE calendar
- Fixed CN holidays
- Make MarketCalendars pickleable

3.2 (10/10/21)

- Major refactoring of the underlying code from PR #150 thanks to https://github.com/Stryder-Git
- Fixed 12/24/1999 early close on NYSE

3.1 (08/29/21)

- Added September 11 holidays to TSX calendar
- Made the minimum version for exchange_calendars >= 3.3 to resolve problem with newer versions of pandas

3.0 (8/17/21)

- Major update to the date_range() functionality. This new behavior is more complete and consistent, but changes behavior in some cases, so a new major version is warranted. For more discussion on the topic refer to PR #142 and Issue #138

2.1 (8/16/21)

- Updated to work with pandas 1.3
- Raise minimum python to 3.7
- NYSE calendar valid from 1885 to present. Includes all full day closes, early closes, and late opens. PR #141

2.0.1 (5/20/21)

- Fixed the TSE calendar for Christmas falling on a Saturday

2.0 (5/8/21)

This version replaces the trading_calendars integration with exchange_calendars, closing out #120. exchange_calendars if the fork of trading_calendars that is currently actively maintained. trading_calendars is now abandoned because it's corporate sponsor is out of business and gone.

1.7 (5/6/21)

This version eliminated the generic CMEExchangeCalendar. This calendar did not represent a specific market and thus was not appropriate for any use. With the addition of the specific calendars for product types this is no longer needed and is removed. To see the product specific calendars here: https://pandas-market-calendars.readthedocs.io/en/latest/calendars.html#futures-calendars

For the CMEEquityExchangeCalendar, this no longer is a mirror of the NYSE calendar as some of the holidays for the NYSE are an open day with early close for CME. This calendar now has its own set of holiday assumptions. This may cause some holidays missing until this calendar is fully tested and vetted.

1.6.2 (5/6/21)

- Fix UK Holidays for #130
- Fix CME Bond calendar for Good Friday #132

1.6.1 (11/3/20)

- Add trading breaks to the trading_calendars import mirror
- Fix the CFE calendar for Good Friday #116
- Renamed XBOM to BSE to avoid conflict with trading_calendars

1.6 (9/14/20)

This is the first version of the merge of this project with the quantopian trading-calendars.

- Added the trading_calendars.py module that brings in all current and future calendars from the quantopian project
- All calendars from trading-calendars are now available in pandas_market_calendars

1.5 (8/30/20)

- Add the is_open_now() function
- Add TASE calendar from #114
- Holiday calendar is now cached to improve performance #117

1.4.2 (8/11/20)

- Fixed for changes to pandas 1.1.0

1.4.1 (7/22/20)

- Added CME_Bond calendar for bond and interest rate futures
- Added futures specific items to the documentations along with examples with breaks

1.4 (7/11/20)

- Add the concept of a break during the trading day. For example this can accommodate Asian markets that have a lunch break, or futures markets that are open 24 hours with a break in the day for trade processing.
- Added product specific contract calendars for CME futures exchange. First calendars are the CME Agricultural and CME Equity calendars
- Add ability to set time zone on schedule() function #42
- Add the Bombay exchange (XBOM) from #96
- Fixed Christmas holidays in SIX #100

1.3 (4/23/20)

- Fixes to support Pandas v1.0
- Remove support for Python 3.4 based on underlying packages removing support for v3.4
- Added ASXExchangeCalendar from PR #85
- Fixes to UK holidays in #84

1.2 (10/22/19)

- Support calendars with valid business days on the weekend (PR #75)
- Fixed SSE 2019 labour's day holidays (PR #74)
- Better JPX calendar support for the time period 1949-2099 (PR #72)
- Reformat Japan's Ascension days, removed duplicate days (PR #68)
- Added German national holidays (PR #77)

1.1 (5/3/19)

- add JPX Ascension Day holidays for 2019 from PR #64

1.0 (3/26/19)

- Official move to Python3 only support
- Version moved to 1.0 as the package has been around and stable long enough to warrant a 1.0

0.22 (3/25/19)

- Added Shanghai Stock Exchange (SSE) calendar from PR #58
- Added HKEX calendar from PR #61
- Fixed tests for pandas v0.24 and higher

0.21 (12/2/18)

- Added Oslo Stock Exchange (OSE) calendar
- Added GW Bush Holiday to NYSE calendar from PR #53 and #54

0.20 (7/2/18)

- Improvements in the internals for how calendars are registered and aliased thanks for PR #45

0.19 (7/2/18)

- schedule() method no longer raises exception if there are no valid trading days between start_date and end_date, will now return an empty DataFrame

0.18 (6/8/18)

- Changed NYSE holiday calendar to start 1/1/1900 (was previously 1/1/1970).
- Fixed an error that schedule() method would fail if the end date was prior to 1993

0.17 (5/24/18)

- Added SIX (Swiss Exchange) calendar, Pull Request #36

0.16 (5/12/18)

- Fixed the equinox for Japanese calendar, Pull Request #33
- Fixed Victoria Day for TSX, issue #34

0.15 (2/23/18)

- Removed toolz as a required package and removed from the one test that required it
- Added daily closes on NYSE back to 1928 from PR #30 thanks to @pldrouin

0.14 (1/7/18)

- Made default open and close times time-zone aware

0.13 (1/5/18)

- Corrected JPX calendar for issue #22

0.12 (12/10/17)

- Added new JPX calendar thanks to gabalese from PR #21

0.11 (10/30/17)

- Corrected the NYSE calendar for Independence Day on Thursday post 2013 to fix #20
- Added new convert_freq() function to convert a date_range to a lower frequency to fix #19

0.10 (9/12/17)

- Added open_time_default and close_time_default as abstract property methods to fix #17

0.9 (9/12/17)

- Fix #12 to Eurex calendar

0.8 (8/24/17)

- Fix #10 to make merge_schedules work properly for more than 2 markets

0.7 (5/30/17)

- Fix a couple deprecated imports

0.6 (3/31/17)

- Added coveralls.io test coverage

0.5 (3/27/17)

- Added Python2.7 support

0.4

- Fixed bug #5

0.3

- Added Eurex calendar

0.2

- Fix to allow start_date and end_date to be the same in schedule()

0.1

- Initial version

CHAPTER 9

Markets & Exchanges

9.1 Calendar Status

9.1.1 Equity Market Calendars

Type	Name	Class	Unit Tests	Creator
Exchange	NYSE	NYSEExchangeCalendar	Yes	Quantopian
Exchange	LSE	LSEExchangeCalendar	Yes	Quantopian
Exchange	CME	CMEExchangeCalendar	Yes	Quantopian
Exchange	ICE	ICEExchangeCalendar	Yes	Quantopian
Exchange	CFE	CFEExchangeCalendar	Yes	Quantopian
Exchange	BMF	BMFExchangeCalendar		Quantopian
Exchange	TSX	TSXExchangeCalendar	Yes	Quantopian
Exchange	EUREX	EUREXExchangeCalendar	Yes	kewlfft
Exchange	JPX	JPXExchangeCalendar	Yes	gabalese
Exchange	SIX	SIXExchangeCalendar	Yes	oliverfu89
Exchange	OSE	OSEExchangeCalendar	Yes	busteren
Exchange	SSE	SSEExchangeCalendar	Yes	keli
Exchange	TASE	TASEExchangeCalendar		gabglus
Exchange	HKEX	HKEXExchangeCalendar	Yes	1dot75cm
Exchange	ASX	ASXExchangeCalendar		pulledlamb
Exchange	BSE	BSEExchangeCalendar		rakesh1988

9.1.2 Futures Calendars

Exchange	Name	Class	Unit Tests	Creator
CME	CME_Equity	CMEEquityExchangeCalendar	Yes	rsheftel
CME	CME_Bond	CMEBondExchangeCalendar	Yes	rsheftel
CME	CME_Agricultural	CMEAgriculturalExchangeCalendar	Yes	lionelyoung
CME	CME Globex Crypto	CMEGlobexCryptoExchangeCalendar	Yes	Coinbase Asset Management

9.1.3 Bond Market Calendars

Country	Name	Class	Unit Tests	Creator
US	SIFMAUS	SIFMAUSExchangeCalendar	Yes	
UK	SIFMAUK	SIFMAUKExchangeCalendar	Yes	
JP	SIFMAJP	SIFMAJPExchangeCalendar	Yes	

9.1.4 Exchange Calendars Package

pandas_market_calendars now imports and provides access to all the calendars in exchange_calendars

Use the ISO code on the trading_calendars page for those calendars. Many of the calendars are duplicated between the pandas_market_calendars and trading_calendars projects. Use whichever one you prefer.

CHAPTER 10

Package Contents

10.1 pandas_market_calendars

10.1.1 pandas_market_calendars package

Submodules

pandas_market_calendars.calendar_registry module

pandas_market_calendars.calendar_registry.**get_calendar**(*name*, *open_time=None*, *close_time=None*) → pandas_market_calendars.market_calendar.MarketCalen

Retrieves an instance of an MarketCalendar whose name is given.

Parameters

- **name** – The name of the MarketCalendar to be retrieved.
- **open_time** – Market open time override as datetime.time object. If None then default is used.
- **close_time** – Market close time override as datetime.time object. If None then default is used.

Returns MarketCalendar of the desired calendar.

pandas_market_calendars.calendar_registry.**get_calendar_names**()

All Market Calendar names and aliases that can be used in "factory" :return: list(str)

pandas_market_calendars.calendar_registry.**times**

Local break end time(s).

As *open_times* although times represent the open of the afternoon subsession. None if exchange does not observe a break.

pandas_market_calendars.calendar_utils module

Utilities to use with market_calendars

pandas_market_calendars.calendar_utils.**convert_freq**(*index*, *frequency*)
Converts a DateTimeIndex to a new lower frequency

Parameters

- **index** – DateTimeIndex
- **frequency** – frequency string

Returns DateTimeIndex

pandas_market_calendars.calendar_utils.**merge_schedules**(*schedules*, *how='outer'*)
Given a list of schedules will return a merged schedule. The merge method (how) will either return the superset of any datetime when any schedule is open (outer) or only the datetime where all markets are open (inner)

CAVEATS:

- This does not work for schedules with breaks, the break information will be lost.
- Only "market_open" and "market_close" are considered, other market times are not yet supported.

Parameters

- **schedules** – list of schedules
- **how** – outer or inner

Returns schedule DataFrame

pandas_market_calendars.class_registry module

class pandas_market_calendars.class_registry.**ProtectedDict**(**args*, ***kwargs*)
Bases: dict

copy() → a shallow copy of D

class pandas_market_calendars.class_registry.**RegisteryMeta**(*name*, *bases*, *attr*)
Bases: type

Metaclass used to register all classes inheriting from RegisteryMeta

pandas_market_calendars.exchange_calendar_asx module

pandas_market_calendars.exchange_calendar_bmf module

pandas_market_calendars.exchange_calendar_cfe module

pandas_market_calendars.exchange_calendar_cme module

pandas_market_calendars.exchange_calendar_eurex module

pandas_market_calendars.exchange_calendar_hkex module

pandas_market_calendars.exchange_calendar_ice module

class pandas_market_calendars.market_calendar.**DEFAULT**
Bases: object

class pandas_market_calendars.market_calendar.**MarketCalendar**(*open_time=None*, *close_time=None*)
Bases: object

An MarketCalendar represents the timing information of a single market or exchange. Unless otherwise noted all times are in UTC and use Pandas data structures.

Parameters

- **open_time** – Market open time override as datetime.time object. If None then default is used.
- **close_time** – Market close time override as datetime.time object. If None then default is used.

add_time(*market_time*, *times*, *opens=<class 'pandas_market_calendars.market_calendar.DEFAULT'>*)
Adds the specified market time to regular_market_times and makes the necessary adjustments.

Parameters

- **market_time** – the market_time to add
- **times** – the time information

- **opens** – see .change_time docstring

Returns None

adhoc_holidays

Returns list of ad-hoc holidays

break_end
Break time end. If None then there is no break

Returns time or None

break_end_on(*date*)

break_start
Break time start. If None then there is no break

Returns time or None

break_start_on(*date*)

classmethod calendar_names()
All Market Calendar names and aliases that can be used in "factory" :return: list(str)

change_time(*market_time*, *times*, *opens=<class 'pandas_market_calendars.market_calendar.DEFAULT'>*)
Changes the specified market time in regular_market_times and makes the necessary adjustments.

Parameters

- **market_time** – the market_time to change
- **times** – new time information
- **opens** – whether the market_time is a time that closes or opens the market this is only needed if the market_time should be respected by .open_at_time True: opens False: closes None: consider it neither opening nor closing, don't add to open_close_map (ignore in .open_at_time) DEFAULT: same as None, unless the market_time is in self.__class__.open_close_map. Then it will take

  the default value as defined by the class.

Returns None

clean_dates(*start_date*, *end_date*)
Strips the inputs of time and time zone information

Parameters

- **start_date** – start date
- **end_date** – end date

Returns (start_date, end_date) with just date, no time and no time zone

close_offset

Returns close offset

close_time
Default close time for the market

Returns time

close_time_on(*date*)

days_at_time(*days*, *market_time*, *day_offset=0*)

Create an index of days at time `t`, interpreted in timezone `tz`. The returned index is localized to UTC.

In the example below, the times switch from 13:45 to 12:45 UTC because March 13th is the daylight savings transition for US/Eastern. All the times are still 8:45 when interpreted in US/Eastern.

```
>>> import pandas as pd; import datetime; import pprint
>>> dts = pd.date_range('2016-03-12', '2016-03-14')
>>> dts_at_845 = days_at_time(dts, datetime.time(8, 45), 'US/Eastern')
>>> pprint.pprint([str(dt) for dt in dts_at_845])
['2016-03-12 13:45:00+00:00',
 '2016-03-13 12:45:00+00:00',
 '2016-03-14 12:45:00+00:00']
```

Parameters

- **days** – DatetimeIndex An index of dates (represented as midnight).
- **market_time** – datetime.time The time to apply as an offset to each day in `days`.
- **day_offset** – int The number of days we want to offset @days by

Returns pd.Series of date with the time requested.

early_closes(*schedule*)

Get a DataFrame of the dates that are an early close.

Parameters **schedule** – schedule DataFrame

Returns schedule DataFrame with rows that are early closes

classmethod factory(*name*, **args*, ***kwargs*)

Parameters

- **cls(RegisteryMeta)** – registration meta class
- **name(str)** – name of class that needs to be instantiated
- **args(Optional(tuple))** – instance positional arguments
- **kwargs(Optional(dict))** – instance named arguments

Returns class instance

get_offset(*market_time*)

get_special_times(*market_time*)

get_special_times_adhoc(*market_time*)

get_time(*market_time*, *all_times=False*)

get_time_on(*market_time*, *date*)

has_custom

has_discontinued

holidays()

Returns the complete CustomBusinessDay object of holidays that can be used in any Pandas function that take that input.

Returns CustomBusinessDay object of holidays

interruptions
This needs to be a list with a tuple for each date that had an interruption. The tuple should have this layout:

(date, start_time, end_time[, start_time2, end_time2, ...])

E.g.: [

("2002-02-03", (time(11), -1), time(11, 2)), ("2010-01-11", time(11), (time(11, 1), 1)), ("2010-01-13", time(9, 59), time(10), time(10, 29), time(10, 30)), ("2011-01-10", time(11), time(11, 1))

]

The date needs to be a string in this format: 'yyyy-mm-dd'. Times need to be two datetime.time objects for each interruption, indicating start and end.

Optionally these can be wrapped in a tuple, where the second element needs to be an integer indicating an offset.

On "2010-01-13" in the example, it is shown that there can be multiple interruptions in a day.

interruptions_df
Will return a pd.DataFrame only containing interruptions.

is_custom(*market_time*)

is_different(*col*, *diff=None*)

is_discontinued(*market_time*)

is_open_now(*schedule*, *include_close=False*, *only_rth=False*)
To determine if the current local system time (converted to UTC) is an open time for the market

Parameters

- **schedule** – schedule DataFrame
- **include_close** – if False then the function will return False if the current local system time is equal to the closing timestamp. If True then it will return True if the current local system time is equal to the closing timestamp. Use True if using bars and would like to include the last bar as a valid open date and time.
- **only_rth** – whether to consider columns that are before market_open or after market_close

Returns True if the current local system time is a valid open date and time, False if not

late_opens(*schedule*)
Get a DataFrame of the dates that are an late opens.

Parameters **schedule** – schedule DataFrame

Returns schedule DataFrame with rows that are late opens

market_times

name
Name of the market

Returns string name

open_at_time(*schedule*, *timestamp*, *include_close=False*, *only_rth=False*)
Determine if a given timestamp is during an open time for the market. If the timestamp is before the first open time or after the last close time of *schedule*, a ValueError will be raised.

Parameters

- **schedule** – schedule DataFrame
- **timestamp** – the timestamp to check for. Assumed to be UTC, if it doesn't include tz information.
- **include_close** – if False then the timestamp that equals the closing timestamp will return False and not be considered a valid open date and time. If True then it will be considered valid and return True. Use True if using bars and would like to include the last bar as a valid open date and time. The close refers to the latest market_time available, which could be after market_close (e.g. 'post').
- **only_rth** – whether to ignore columns that are before market_open or after market_close. If true, include_close will be referring to market_close.

Returns True if the timestamp is a valid open date and time, False if not

open_close_map = {'break_end': True, 'break_start': False, 'market_close': False, '

open_offset

Returns open offset

open_time
Default open time for the market

Returns time

open_time_on(*date*)

regular_holidays

Returns pd.AbstractHolidayCalendar: a calendar containing the regular holidays for this calendar

regular_market_times = {'market_close': ((None, datetime.time(23, 0)),), 'market_open

remove_time(*market_time*)
Removes the specified market time from regular_market_times and makes the necessary adjustments.

Parameters **market_time** – the market_time to remove

Returns None

schedule(*start_date*, *end_date*, *tz='UTC'*, *start='market_open'*, *end='market_close'*, *force_special_times=True*, *market_times=None*, *interruptions=False*)
Generates the schedule DataFrame. The resulting DataFrame will have all the valid business days as the index and columns for the requested market times. The columns can be determined either by setting a range (inclusive on both sides), using *start* and *end*, or by passing a list to 'market_times'. A range of market_times is derived from a list of market_times that are available to the instance, which are sorted based on the current regular time. See examples/usage.ipynb for demonstrations.

All time zones are set to UTC by default. Setting the tz parameter will convert the columns to the desired timezone, such as 'America/New_York'.

Parameters

- **start_date** – first date of the schedule
- **end_date** – last date of the schedule
- **tz** – timezone that the columns of the returned schedule are in, default: "UTC"
- **start** – the first market_time to include as a column, default: "market_open"
- **end** – the last market_time to include as a column, default: "market_close"

- **force_special_times** – how to handle special times. True: overwrite regular times of the column itself, conform other columns to special times of

  market_open/market_close if those are requested.

  False: only overwrite regular times of the column itself, leave others alone None: completely ignore special times

- **market_times** – alternative to start/end, list of market_times that are in self.regular_market_times
- **interruptions** – bool, whether to add interruptions to the schedule, default: False These will be added as columns to the right of the DataFrame. Any interruption on a day between start_date and end_date will be included, regardless of the market_times requested. Also, *force_special_times* does not take these into consideration.

Returns schedule DataFrame

special_closes
A list of special close times and corresponding HolidayCalendars.

Returns List of (time, AbstractHolidayCalendar) tuples

special_closes_adhoc

Returns List of (time, DatetimeIndex) tuples that represent special closes that cannot be codified into rules.

special_dates(*market_time*, *start_date*, *end_date*, *filter_holidays=True*)
Calculate a datetimeindex that only contains the specail times of the requested market time.

Parameters

- **market_time** – market_time reference
- **start_date** – first possible date of the index
- **end_date** – last possible date of the index
- **filter_holidays** – will filter days by self.valid_days, which can be useful when debugging

Returns schedule DatetimeIndex

special_market_close
A list of special close times and corresponding HolidayCalendars.

Returns List of (time, AbstractHolidayCalendar) tuples

special_market_close_adhoc

Returns List of (time, DatetimeIndex) tuples that represent special closes that cannot be codified into rules.

special_market_open
A list of special open times and corresponding AbstractHolidayCalendar.

Returns List of (time, AbstractHolidayCalendar) tuples

special_market_open_adhoc

Returns List of (time, DatetimeIndex) tuples that represent special opens that cannot be codified into rules.

special_opens
A list of special open times and corresponding AbstractHolidayCalendar.

Returns List of (time, AbstractHolidayCalendar) tuples

special_opens_adhoc

Returns List of (time, DatetimeIndex) tuples that represent special opens that cannot be codified into rules.

tz

Time zone for the market.

Returns timezone

valid_days(*start_date*, *end_date*, *tz='UTC'*)

Get a DatetimeIndex of valid open business days.

Parameters

- **start_date** – start date
- **end_date** – end date
- **tz** – time zone in either string or pytz.timezone

Returns DatetimeIndex of valid business days

weekmask

class pandas_market_calendars.market_calendar.**MarketCalendarMeta**(*name*, *bases*, *attr*)

Bases: abc.ABCMeta, *pandas_market_calendars.class_registry.RegisteryMeta*

pandas_market_calendars.trading_calendars_mirror module

Module contents

class pandas_market_calendars.**MarketCalendar**(*open_time=None*, *close_time=None*)

Bases: object

An MarketCalendar represents the timing information of a single market or exchange. Unless otherwise noted all times are in UTC and use Pandas data structures.

Parameters

- **open_time** – Market open time override as datetime.time object. If None then default is used.
- **close_time** – Market close time override as datetime.time object. If None then default is used.

add_time(*market_time*, *times*, *opens=<class 'pandas_market_calendars.market_calendar.DEFAULT'>*)

Adds the specified market time to regular_market_times and makes the necessary adjustments.

Parameters

- **market_time** – the market_time to add
- **times** – the time information
- **opens** – see .change_time docstring

Returns None

adhoc_holidays

Returns list of ad-hoc holidays

break_end
Break time end. If None then there is no break

Returns time or None

break_end_on(*date*)

break_start
Break time start. If None then there is no break

Returns time or None

break_start_on(*date*)

classmethod calendar_names()
All Market Calendar names and aliases that can be used in "factory" :return: list(str)

change_time(*market_time*, *times*, *opens=<class 'pandas_market_calendars.market_calendar.DEFAULT'>*)
Changes the specified market time in regular_market_times and makes the necessary adjustments.

Parameters

- **market_time** – the market_time to change
- **times** – new time information
- **opens** – whether the market_time is a time that closes or opens the market this is only needed if the market_time should be respected by .open_at_time True: opens False: closes None: consider it neither opening nor closing, don't add to open_close_map (ignore in .open_at_time) DEFAULT: same as None, unless the market_time is in self.__class__.open_close_map. Then it will take

  the default value as defined by the class.

Returns None

clean_dates(*start_date*, *end_date*)
Strips the inputs of time and time zone information

Parameters

- **start_date** – start date
- **end_date** – end date

Returns (start_date, end_date) with just date, no time and no time zone

close_offset

Returns close offset

close_time
Default close time for the market

Returns time

close_time_on(*date*)

days_at_time(*days*, *market_time*, *day_offset=0*)
Create an index of days at time `t`, interpreted in timezone `tz`. The returned index is localized to UTC.

In the example below, the times switch from 13:45 to 12:45 UTC because March 13th is the daylight savings transition for US/Eastern. All the times are still 8:45 when interpreted in US/Eastern.

```
>>> import pandas as pd; import datetime; import pprint
>>> dts = pd.date_range('2016-03-12', '2016-03-14')
>>> dts_at_845 = days_at_time(dts, datetime.time(8, 45), 'US/Eastern')
>>> pprint.pprint([str(dt) for dt in dts_at_845])
['2016-03-12 13:45:00+00:00',
 '2016-03-13 12:45:00+00:00',
 '2016-03-14 12:45:00+00:00']
```

Parameters

- **days** – DatetimeIndex An index of dates (represented as midnight).
- **market_time** – datetime.time The time to apply as an offset to each day in `days`.
- **day_offset** – int The number of days we want to offset @days by

Returns pd.Series of date with the time requested.

early_closes(*schedule*)

Get a DataFrame of the dates that are an early close.

Parameters **schedule** – schedule DataFrame

Returns schedule DataFrame with rows that are early closes

classmethod factory(*name*, **args*, ***kwargs*)

Parameters

- **cls(RegisteryMeta)** – registration meta class
- **name(str)** – name of class that needs to be instantiated
- **args(Optional(tuple))** – instance positional arguments
- **kwargs(Optional(dict))** – instance named arguments

Returns class instance

get_offset(*market_time*)

get_special_times(*market_time*)

get_special_times_adhoc(*market_time*)

get_time(*market_time*, *all_times=False*)

get_time_on(*market_time*, *date*)

has_custom

has_discontinued

holidays()

Returns the complete CustomBusinessDay object of holidays that can be used in any Pandas function that take that input.

Returns CustomBusinessDay object of holidays

interruptions

This needs to be a list with a tuple for each date that had an interruption. The tuple should have this layout:

(date, start_time, end_time[, start_time2, end_time2, ...])

E.g.: [

("2002-02-03", (time(11), -1), time(11, 2)), ("2010-01-11", time(11), (time(11, 1), 1)), ("2010-01-13", time(9, 59), time(10), time(10, 29), time(10, 30)), ("2011-01-10", time(11), time(11, 1))

]

The date needs to be a string in this format: 'yyyy-mm-dd'. Times need to be two datetime.time objects for each interruption, indicating start and end.

Optionally these can be wrapped in a tuple, where the second element needs to be an integer indicating an offset.

On "2010-01-13" in the example, it is shown that there can be multiple interruptions in a day.

interruptions_df
Will return a pd.DataFrame only containing interruptions.

is_custom(*market_time*)

is_different(*col*, *diff=None*)

is_discontinued(*market_time*)

is_open_now(*schedule*, *include_close=False*, *only_rth=False*)
To determine if the current local system time (converted to UTC) is an open time for the market

Parameters

- **schedule** – schedule DataFrame
- **include_close** – if False then the function will return False if the current local system time is equal to the closing timestamp. If True then it will return True if the current local system time is equal to the closing timestamp. Use True if using bars and would like to include the last bar as a valid open date and time.
- **only_rth** – whether to consider columns that are before market_open or after market_close

Returns True if the current local system time is a valid open date and time, False if not

late_opens(*schedule*)
Get a DataFrame of the dates that are an late opens.

Parameters **schedule** – schedule DataFrame

Returns schedule DataFrame with rows that are late opens

market_times

name
Name of the market

Returns string name

open_at_time(*schedule*, *timestamp*, *include_close=False*, *only_rth=False*)
Determine if a given timestamp is during an open time for the market. If the timestamp is before the first open time or after the last close time of *schedule*, a ValueError will be raised.

Parameters

- **schedule** – schedule DataFrame
- **timestamp** – the timestamp to check for. Assumed to be UTC, if it doesn't include tz information.

- **include_close** – if False then the timestamp that equals the closing timestamp will return False and not be considered a valid open date and time. If True then it will be considered valid and return True. Use True if using bars and would like to include the last bar as a valid open date and time. The close refers to the latest market_time available, which could be after market_close (e.g. 'post').
- **only_rth** – whether to ignore columns that are before market_open or after market_close. If true, include_close will be referring to market_close.

Returns True if the timestamp is a valid open date and time, False if not

open_close_map = {'break_end': True, 'break_start': False, 'market_close': False, '

open_offset

Returns open offset

open_time
Default open time for the market

Returns time

open_time_on(*date*)

regular_holidays

Returns pd.AbstractHolidayCalendar: a calendar containing the regular holidays for this calendar

regular_market_times = {'market_close': ((None, datetime.time(23, 0)),), 'market_open

remove_time(*market_time*)
Removes the specified market time from regular_market_times and makes the necessary adjustments.

Parameters **market_time** – the market_time to remove

Returns None

schedule(*start_date*, *end_date*, *tz='UTC'*, *start='market_open'*, *end='market_close'*, *force_special_times=True*, *market_times=None*, *interruptions=False*)
Generates the schedule DataFrame. The resulting DataFrame will have all the valid business days as the index and columns for the requested market times. The columns can be determined either by setting a range (inclusive on both sides), using *start* and *end*, or by passing a list to 'market_times'. A range of market_times is derived from a list of market_times that are available to the instance, which are sorted based on the current regular time. See examples/usage.ipynb for demonstrations.

All time zones are set to UTC by default. Setting the tz parameter will convert the columns to the desired timezone, such as 'America/New_York'.

Parameters

- **start_date** – first date of the schedule
- **end_date** – last date of the schedule
- **tz** – timezone that the columns of the returned schedule are in, default: "UTC"
- **start** – the first market_time to include as a column, default: "market_open"
- **end** – the last market_time to include as a column, default: "market_close"
- **force_special_times** – how to handle special times. True: overwrite regular times of the column itself, conform other columns to special times of

  market_open/market_close if those are requested.

False: only overwrite regular times of the column itself, leave others alone None: completely ignore special times

- **market_times** – alternative to start/end, list of market_times that are in self.regular_market_times
- **interruptions** – bool, whether to add interruptions to the schedule, default: False These will be added as columns to the right of the DataFrame. Any interruption on a day between start_date and end_date will be included, regardless of the market_times requested. Also, *force_special_times* does not take these into consideration.

Returns schedule DataFrame

special_closes
A list of special close times and corresponding HolidayCalendars.

Returns List of (time, AbstractHolidayCalendar) tuples

special_closes_adhoc

Returns List of (time, DatetimeIndex) tuples that represent special closes that cannot be codified into rules.

special_dates(*market_time*, *start_date*, *end_date*, *filter_holidays=True*)
Calculate a datetimeindex that only contains the specail times of the requested market time.

Parameters

- **market_time** – market_time reference
- **start_date** – first possible date of the index
- **end_date** – last possible date of the index
- **filter_holidays** – will filter days by self.valid_days, which can be useful when debugging

Returns schedule DatetimeIndex

special_market_close
A list of special close times and corresponding HolidayCalendars.

Returns List of (time, AbstractHolidayCalendar) tuples

special_market_close_adhoc

Returns List of (time, DatetimeIndex) tuples that represent special closes that cannot be codified into rules.

special_market_open
A list of special open times and corresponding AbstractHolidayCalendar.

Returns List of (time, AbstractHolidayCalendar) tuples

special_market_open_adhoc

Returns List of (time, DatetimeIndex) tuples that represent special opens that cannot be codified into rules.

special_opens
A list of special open times and corresponding AbstractHolidayCalendar.

Returns List of (time, AbstractHolidayCalendar) tuples

special_opens_adhoc

Returns List of (time, DatetimeIndex) tuples that represent special opens that cannot be codified into rules.

tz
Time zone for the market.

Returns timezone

valid_days(*start_date*, *end_date*, *tz='UTC'*)
Get a DatetimeIndex of valid open business days.

Parameters

- **start_date** – start date
- **end_date** – end date
- **tz** – time zone in either string or pytz.timezone

Returns DatetimeIndex of valid business days

weekmask

pandas_market_calendars.**get_calendar**(*name*, *open_time=None*, *close_time=None*) → pandas_market_calendars.market_calendar.MarketCalendar
Retrieves an instance of an MarketCalendar whose name is given.

Parameters

- **name** – The name of the MarketCalendar to be retrieved.
- **open_time** – Market open time override as datetime.time object. If None then default is used.
- **close_time** – Market close time override as datetime.time object. If None then default is used.

Returns MarketCalendar of the desired calendar.

pandas_market_calendars.**get_calendar_names**()
All Market Calendar names and aliases that can be used in "factory" :return: list(str)

pandas_market_calendars.**merge_schedules**(*schedules*, *how='outer'*)
Given a list of schedules will return a merged schedule. The merge method (how) will either return the superset of any datetime when any schedule is open (outer) or only the datetime where all markets are open (inner)

CAVEATS:

- This does not work for schedules with breaks, the break information will be lost.
- Only "market_open" and "market_close" are considered, other market times are not yet supported.

Parameters

- **schedules** – list of schedules
- **how** – outer or inner

Returns schedule DataFrame

pandas_market_calendars.**convert_freq**(*index*, *frequency*)
Converts a DateTimeIndex to a new lower frequency

Parameters

- **index** – DateTimeIndex

- **frequency** – frequency string

Returns DateTimeIndex

CHAPTER 11

Examples

```
import sys
sys.path.append("../")
from datetime import time
import pandas as pd
import pandas_market_calendars as mcal
```

11.1 Calendars

11.1.1 Basic Usage

Setup new exchange calendar

```
nyse = mcal.get_calendar('NYSE')
```

Get the time zone

```
nyse.tz.zone
```

```
'America/New_York'
```

Get the AbstractHolidayCalendar object

```
holidays = nyse.holidays()
holidays.holidays[-5:]
```

```
(numpy.datetime64('2200-06-19'),
 numpy.datetime64('2200-07-04'),
 numpy.datetime64('2200-09-01'),
 numpy.datetime64('2200-11-27'),
 numpy.datetime64('2200-12-25'))
```

View the available information on regular market times

```
print(nyse.regular_market_times) # more on this under the 'Customizations' heading
```

```
ProtectedDict(
{'pre': ((None, datetime.time(4, 0)),),
 'market_open': ((None, datetime.time(10, 0)),
                 ('1985-01-01', datetime.time(9, 30))),
 'market_close': ((None, datetime.time(15, 0)),
                  ('1952-09-29', datetime.time(15, 30)),
                  ('1974-01-01', datetime.time(16, 0))),
 'post': ((None, datetime.time(20, 0)),)}
)
```

Exchange open valid business days

Get the valid open exchange business dates between a start and end date. Note that Dec 26 (Christmas), Jan 2 (New Years) and all weekends are missing

```
nyse.valid_days(start_date='2016-12-20', end_date='2017-01-10')
```

```
DatetimeIndex(['2016-12-20 00:00:00+00:00', '2016-12-21 00:00:00+00:00',
               '2016-12-22 00:00:00+00:00', '2016-12-23 00:00:00+00:00',
               '2016-12-27 00:00:00+00:00', '2016-12-28 00:00:00+00:00',
               '2016-12-29 00:00:00+00:00', '2016-12-30 00:00:00+00:00',
               '2017-01-03 00:00:00+00:00', '2017-01-04 00:00:00+00:00',
               '2017-01-05 00:00:00+00:00', '2017-01-06 00:00:00+00:00',
               '2017-01-09 00:00:00+00:00', '2017-01-10 00:00:00+00:00'],
              dtype='datetime64[ns, UTC]', freq=None)
```

Schedule

```
schedule = nyse.schedule(start_date='2016-12-30', end_date='2017-01-10')
schedule
```

```
# with early closes
early = nyse.schedule(start_date='2012-07-01', end_date='2012-07-10')
early
```

```
# including pre and post-market
extended = nyse.schedule(start_date='2012-07-01', end_date='2012-07-10', start="pre",␣
↪end="post")
extended
```

```
# specific market times
# CAVEAT: Looking at 2012-07-03, you can see that times will NOT be adjusted to␣
↪special_opens/sepcial_closes
# if market_open/market_close are not requested
specific = nyse.schedule(start_date='2012-07-01', end_date='2012-07-10', market_
↪times= ["post", "market_open"]) # this order will be kept
specific
```

Get early closes

```
nyse.early_closes(schedule=early)
```

```
nyse.early_closes(schedule=extended)
```

Open at time

Test to see if a given timestamp is during market open hours. (You can find more on this under the 'Advanced open_at_time' header)

```
nyse.open_at_time(early, pd.Timestamp('2012-07-03 12:00', tz='America/New_York'))
```

```
True
```

```
nyse.open_at_time(early, pd.Timestamp('2012-07-03 16:00', tz='America/New_York'))
```

```
False
```

Other market times will also be considered

```
nyse.open_at_time(extended, pd.Timestamp('2012-07-05 18:00', tz='America/New_York'))
```

```
True
```

but can be ignored by setting only_rth = True

```
nyse.open_at_time(extended, pd.Timestamp('2012-07-05 18:00', tz='America/New_York'),␣
↪only_rth = True)
```

```
False
```

11.2 Customizations

The simplest way to customize the market times of a calendar is by passing datetime.time objects to the constructor, which will modify the open and/or close of *regular trading hours*.

```
cal = mcal.get_calendar('NYSE', open_time=time(10, 0), close_time=time(14, 30))
print('open, close: %s, %s' % (cal.open_time, cal.close_time))
```

```
open, close: 10:00:00, 14:30:00
```

More advanced customizations can be done after initialization or by inheriting from the closest MarketCalendar class, which requires an explanation of market times...

11.2.1 Market times

Market times are moments in a trading day that are contained in the `regular_market_times` attribute, for example:

```
print("The original NYSE calendar: \n", nyse.regular_market_times)
```

```
The original NYSE calendar:
 ProtectedDict(
{'pre': ((None, datetime.time(4, 0)),),
 'market_open': ((None, datetime.time(10, 0)),
                 ('1985-01-01', datetime.time(9, 30))),
 'market_close': ((None, datetime.time(15, 0)),
                  ('1952-09-29', datetime.time(15, 30)),
                  ('1974-01-01', datetime.time(16, 0))),
 'post': ((None, datetime.time(20, 0)),)}
)
```

NYSE's regular trading hours are referenced by "market_open" and "market_close", but NYSE also has extended hours, which are referenced by "pre" and "post".

The attribute ''regular_market_times'' has these requirements:

- It needs to be a dictionary
- Each market_time needs one entry
  - Regular open must be "market_open", regular close must be "market_close".
  - If there is a break, there must be a "break_start" and a "break_end".
  - only ONE break is currently supported.
- One tuple for each market_time, containing at least one tuple:
  - Each nested tuple needs at least two items: `(first_date_used, time[, offset])`.
  - The first tuple's date should be None, marking the start. In every tuple thereafter this is the date when `time` was first used.
  - Optionally (assumed to be zero, when not present), a positive or negative integer, representing an offset in number of days.
  - Dates need to be in ascending order, None coming first.

E.g.:

```
print(nyse.get_time("market_close", all_times= True)) # all_times = False only␣
↪returns current
```

```
((None, datetime.time(15, 0)), ('1952-09-29', datetime.time(15, 30)), ('1974-01-01',␣
↪datetime.time(16, 0)))
```

The first known close was 3pm, which changed on 1952-09-29 to 3:30pm, which changed on 1974-01-01 to 4pm. The dates are the first dates that the new time was used.

Customizing after initialization

There are three methods that allow customizing the `regular_market_times` of a MarketCalendar instance: * `.change_time(market_time, times)` * `.add_time(market_time, times)` * `.remove_time(market_time)`

```
cal = mcal.get_calendar("NYSE")
cal.change_time("market_open", time(10,30))
print('open, close: %s, %s' % (cal.open_time, cal.close_time))
print("\nThe 'market_open' information is entirely replaced:\n", cal.regular_market_
↪times)
```

```
open, close: 10:30:00, 16:00:00

The 'market_open' information is entirely replaced:
 ProtectedDict(
{'pre': ((None, datetime.time(4, 0)),),
 'market_open': ((None, datetime.time(10, 30)),),
 'market_close': ((None, datetime.time(15, 0)),
                  ('1952-09-29', datetime.time(15, 30)),
                  ('1974-01-01', datetime.time(16, 0))),
 'post': ((None, datetime.time(20, 0)),)}
)
```

```
cal.remove_time("post")
cal.add_time("new_post", time(19))
print(cal.regular_market_times)
```

```
ProtectedDict(
{'pre': ((None, datetime.time(4, 0)),),
 'market_open': ((None, datetime.time(10, 30)),),
 'market_close': ((None, datetime.time(15, 0)),
                  ('1952-09-29', datetime.time(15, 30)),
                  ('1974-01-01', datetime.time(16, 0))),
 'new_post': ((None, datetime.time(19, 0)),)}
)
```

```
cal.remove_time("pre")
cal.remove_time("new_post")
```

The methods `.add_time` and `.change_time` also accept the time information in these formats:

```
cal.add_time("just_time", time(10))
cal.add_time("with_offset", (time(10), -1))
cal.add_time("changes_and_offset", ((None, time(17)), ("2009-12-28", time(11), -2)))
print(cal.regular_market_times)
```

```
ProtectedDict(
{'market_open': ((None, datetime.time(10, 30)),),
 'market_close': ((None, datetime.time(15, 0)),
                  ('1952-09-29', datetime.time(15, 30)),
                  ('1974-01-01', datetime.time(16, 0))),
 'just_time': ((None, datetime.time(10, 0)),),
 'with_offset': ((None, datetime.time(10, 0), -1),),
 'changes_and_offset': ((None, datetime.time(17, 0)),
                        ('2009-12-28', datetime.time(11, 0), -2))}
)
```

CAVEATS:

FIRST

Internally, an order of market_times is detected based on their current time.
Because of the offsets in "with_offset" and "changes_and_offset", the columns in a schedule are in the following order:

```
cal.schedule("2009-12-23", "2009-12-29", market_times= "all")
```

On 2009-12-23 changes_and_offset doesn't seem to be in the right order, but as of 2009-12-28 it is.

Passing a list to `market_times`, allows you to keep a custom order:

```
cal.schedule("2009-12-23", "2009-12-29", market_times= ["with_offset", "market_open",
↪"market_close", "changes_and_offset"])
```

SECOND

Special closes of market_closes will override all later times, special opens of market_opens will override all earlier times.
In the prior schedule, 2009-12-24 is a special market_close, which was enforced in the changes_and_offset column.

Providing `False` or `None` to the `force_special_times` keyword argument, changes this behaviour:

```
# False - will only adjust the columns itself (changes_and_offset left alone, market_
↪close adjusted)
cal.schedule("2009-12-23", "2009-12-28", market_times= ["changes_and_offset", "market_
↪close"], force_special_times= False)
```

```
# None - will not adjust any column (both are left alone)
cal.schedule("2009-12-23", "2009-12-28", market_times= ["changes_and_offset", "market_
↪close"], force_special_times= None)
```

Inheriting from a MarketCalendar

You get even more control over a calendar (or help this package by contributing a calendar) by inheriting from a MarketCalendar class. The following three sections cover:

```
* Setting special times for market_times
* Setting interruptions
* How to make sure open_at_time works
```

Special Times

Any market_time in regular_market_times can have special times, which are looked for in two properties:

```
special_{market_time}_adhoc
    same format as special_opens_adhoc, which is the same as special_market_open_adhoc
special_{market_time}
    same format as special_opens, which is the same as special_market_open
```

```
# For example, CFEExchangeCalendar only has the regular trading hours for the futures␣
↪exchange (8:30 - 15:15).
# If you want to use the equity options exchange (8:30 - 15:00), including the order␣
↪acceptance time at 7:30, and
# some special cases when the order acceptance time was different, do this:

from pandas_market_calendars.exchange_calendar_cboe import CFEExchangeCalendar

class DemoOptionsCalendar(CFEExchangeCalendar):  # Inherit what doesn't need to change
    name = "Demo_Options"
    aliases = [name]
    regular_market_times = {**CFEExchangeCalendar.regular_market_times, # unpack the␣
↪parent's regular_market_times
                            "order_acceptance": ((None, time(7,30)),),  # add your␣
↪market time of interest
                            "market_close": ((None, time(15)),)} # overwrite the␣
↪market time you want to change

    @property
    def special_order_acceptance_adhoc(self):  # include special cases
        return [(time(8,30), ["2000-12-27", "2001-12-27"])]
```

```
options = mcal.get_calendar("Demo_Options")

print(options.regular_market_times)
```

```
ProtectedDict(
{'market_open': ((None, datetime.time(8, 30)),),
 'market_close': ((None, datetime.time(15, 0)),),
 'order_acceptance': ((None, datetime.time(7, 30)),)}
)
```

```
schedule = options.schedule("2000-12-22", "2000-12-28", start= "order_acceptance")
schedule
```

Dec 25th is filtered out already because it is inherited from the CFEExchangeCalendar, and the special case on 2000-12-27 is also integrated

Interruptions

MarketCalendar subclasses also support interruptions, which can be defined in the `interruptions` property. To view interruptions, you can use the `interruptions_df` property or set `interruptions= True` when calling `schedule`.

```
class InterruptionsDemo(DemoOptionsCalendar):
    @property
    def interruptions(self):
        return [
            ("2002-02-03", (time(11), -1), time(11, 2)),
            ("2010-01-11", time(11), (time(11, 1), 1)),
            ("2010-01-13", time(9, 59), time(10), time(10, 29), time(10, 30)),
            ("2011-01-10", time(11), time(11, 1))]
```

```
cal = InterruptionsDemo()
```

```
cal.interruptions_df
```

```
sched = cal.schedule("2010-01-09", "2010-01-15", interruptions= True)
sched
```

```
def is_open(c, s, *dates):
    for t in dates:
        print("open on", t, ":", c.open_at_time(s, t))
```

Advanced open_at_time

`MarketCalendar.open_at_time` uses the class attribute `open_close_map` to determine if a market_time opens or closes the market. It will also look for the 'interruption_' prefix in the columns to respect interruptions.

Here you can see that MarketCalendar.open_at_time respects interruptions (the last two timestamps):

```
is_open(cal, sched, "2010-01-12 14:00:00", "2010-01-12 14:35:00","2010-01-13 15:59:00
↪","2010-01-13 16:30:00")
```

```
open on 2010-01-12 14:00:00 : False
open on 2010-01-12 14:35:00 : True
open on 2010-01-13 15:59:00 : False
open on 2010-01-13 16:30:00 : True
```

In the `DemoOptionsCalendar`, we did not specify what order_acceptance means for the market, which will not allow open_at_time to work.

```
sched = cal.schedule("2010-01-09", "2010-01-15", start= "order_acceptance",␣
↪interruptions= True)
try:
    cal.open_at_time(sched, "2010-01-12")
except ValueError as e:
    print(e)
```

```
You seem to be using a schedule that isn't based on the market_times, or includes␣
↪market_times that are not represented in the open_close_map.
```

```
# These are the defaults that every MarketCalendar has, which is still missing order_
↪accpetance.
print(cal.open_close_map)
```

```
ProtectedDict(
{'market_open': True,
 'market_close': False,
 'break_start': False,
 'break_end': True,
 'pre': True,
 'post': False}
)
```

To correct the calendar we should include the following:

```
class OpenCloseDemo(InterruptionsDemo):

    open_close_map = {**CFEExchangeCalendar.open_close_map,
                      "order_acceptance": True}

cal = OpenCloseDemo()

sched = cal.schedule("2010-01-09", "2010-01-15", start= "order_acceptance",␣
↪interruptions= True)
sched
```

Now we can see that not only interruptions (last two) but also order_acceptance (first) is respected

```
is_open(cal, sched, "2010-01-11 13:35:00", "2010-01-12 14:35:00", "2010-01-13 15:59:00
↪", "2010-01-13 16:30:00")
```

```
open on 2010-01-11 13:35:00 : True
open on 2010-01-12 14:35:00 : True
open on 2010-01-13 15:59:00 : False
open on 2010-01-13 16:30:00 : True
```

You can even change this dynamically, using the `opens` keyword in `.change_time` and `.add_time`

```
cal.change_time("order_acceptance", cal["order_acceptance"], opens= False)

is_open(cal, sched, "2010-01-11 13:35:00", "2010-01-12 14:35:00", "2010-01-13 15:59:00
↪", "2010-01-13 16:30:00")
```

```
open on 2010-01-11 13:35:00 : False
open on 2010-01-12 14:35:00 : True
open on 2010-01-13 15:59:00 : False
open on 2010-01-13 16:30:00 : True
```

```
cal.change_time("order_acceptance", cal["order_acceptance"], opens= True)

cal.add_time("order_closed", time(8), opens= False)

sched = cal.schedule("2010-01-09", "2010-01-15", start= "order_acceptance")
sched
```

```
is_open(cal, sched, "2010-01-11 13:35:00", "2010-01-11 14:15:00", "2010-01-11 14:35:00
↪")
```

```
open on 2010-01-11 13:35:00 : True
open on 2010-01-11 14:15:00 : False
open on 2010-01-11 14:35:00 : True
```

11.3 Extra Usage

11.3.1 Checking for special times

The following functions respect varying times in regular_market_times

These will only check market_close/market_open columns for early/late times

```
options.early_closes(schedule), options.late_opens(schedule)
```

```
(Empty DataFrame
 Columns: [order_acceptance, market_open, market_close]
 Index: [],
 Empty DataFrame
 Columns: [order_acceptance, market_open, market_close]
 Index: [])
```

The `is_different` method uses the name of the series passed to it, to determine which rows are not equal to the regular market times, and return a boolean Series

```
schedule[options.is_different(schedule["order_acceptance"])]
```

You can also pass `pd.Series.lt/ -.gt / -.ge / etc.` for more control over the comparison

```
schedule[options.is_different(schedule["order_acceptance"], pd.Series.lt)]
```

```
schedule[options.is_different(schedule["order_acceptance"], pd.Series.ge)]
```

Checking custom times

```
options.has_custom # order_acceptance is not considered custom because it is␣
↪hardcoded into the class
```

```
False
```

```
options.add_time("post", time(17))
```

```
options.has_custom, options.is_custom("market_open"), options.is_custom("post")
```

```
(True, False, True)
```

Get the regular time on a certain date

```
nyse.open_time, nyse.close_time  # these always refer to the current time of market_
↪open/market_close
```

```
(datetime.time(9, 30, tzinfo=<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>),
 datetime.time(16, 0, tzinfo=<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>))
```

```
nyse.get_time("post"), nyse.get_time("pre")  # these also refer to the current time
```

```
(datetime.time(20, 0, tzinfo=<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>),
 datetime.time(4, 0, tzinfo=<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>))
```

```
# open_time_on looks for market_open, close_time_on looks for market_close and get_
↪time_on looks for the provided market time
nyse.open_time_on("1950-01-01"), nyse.get_time_on("market_close", "1960-01-01")
```

```
(datetime.time(10, 0, tzinfo=<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>),
 datetime.time(15, 30, tzinfo=<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>))
```

Special Methods

```
nyse["market_open"] # gets the current time
```

```
datetime.time(9, 30, tzinfo=<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>)
```

```
nyse["market_open", "all"] # gets all times
```

```
((None, datetime.time(10, 0)), ('1985-01-01', datetime.time(9, 30)))
```

```
nyse["market_open", "1950-01-01"] # gets the time on a certain date
```

```
datetime.time(10, 0, tzinfo=<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>)
```

This tries to *add* a time, which will fail if it already exists. In that case `.change_time` is the explicit alternative.

```
nyse["new_post"] = time(20)
nyse["new_post"]
```

```
datetime.time(20, 0, tzinfo=<DstTzInfo 'America/New_York' LMT-1 day, 19:04:00 STD>)
```

```
try: nyse["post"] = time(19)
except AssertionError as e: print(e)
```

```
post is already in regular_market_times:
['pre', 'market_open', 'market_close', 'post', 'new_post']
```

Array of special times

```
options.special_dates("order_acceptance", "2000-12-22", "2001-12-28")
```

```
2000-12-27   2000-12-27 14:30:00+00:00
2001-12-27   2001-12-27 14:30:00+00:00
dtype: datetime64[ns, UTC]
```

Handling discontinued times

```
xkrx = mcal.get_calendar("XKRX")
```

```
/opt/hostedtoolcache/Python/3.10.9/x64/lib/python3.10/site-packages/pandas_
↪market_calendars/market_calendar.py:144: UserWarning: ['break_start',␣
↪'break_end'] are discontinued, the dictionary .discontinued_market_times␣
↪has the dates on which these were discontinued. The times as of those dates␣
↪are incorrect, use .remove_time(market_time) to ignore a market_time.
  warnings.warn(f"{list(discontinued.keys())} are discontinued, the␣
↪dictionary"
```

```
xkrx.schedule("2020-01-01", "2020-01-05")
```

```
xkrx.discontinued_market_times # these are the dates as of which the market time didn
↪'t exist anymore
```

```
ProtectedDict({'break_start': Timestamp('2000-05-22 00:00:00'), 'break_end':␣
↪Timestamp('2000-05-22 00:00:00')})
```

```
print(xkrx.has_discontinued)
xkrx.remove_time("break_start")
xkrx.remove_time("break_end")
print(xkrx.has_discontinued)
```

```
True
False
```

```
xkrx.schedule("2020-01-01", "2020-01-05")
```

11.4 Helpers

schedules with columns other than market_open, break_start, break_end or market_close are not yet supported by the following functions

11.4.1 Date Range

This function will take a schedule DataFrame and return a DatetimeIndex with all timestamps at the frequency given for all of the exchange open dates and times.

```
mcal.date_range(early, frequency='1D')
```

```
DatetimeIndex(['2012-07-02 20:00:00+00:00', '2012-07-03 17:00:00+00:00',
               '2012-07-05 20:00:00+00:00', '2012-07-06 20:00:00+00:00',
               '2012-07-09 20:00:00+00:00', '2012-07-10 20:00:00+00:00'],
              dtype='datetime64[ns, UTC]', freq=None)
```

```
mcal.date_range(early, frequency='1H')
```

```
DatetimeIndex(['2012-07-02 14:30:00+00:00', '2012-07-02 15:30:00+00:00',
               '2012-07-02 16:30:00+00:00', '2012-07-02 17:30:00+00:00',
               '2012-07-02 18:30:00+00:00', '2012-07-02 19:30:00+00:00',
               '2012-07-02 20:00:00+00:00', '2012-07-03 14:30:00+00:00',
               '2012-07-03 15:30:00+00:00', '2012-07-03 16:30:00+00:00',
               '2012-07-03 17:00:00+00:00', '2012-07-05 14:30:00+00:00',
               '2012-07-05 15:30:00+00:00', '2012-07-05 16:30:00+00:00',
               '2012-07-05 17:30:00+00:00', '2012-07-05 18:30:00+00:00',
               '2012-07-05 19:30:00+00:00', '2012-07-05 20:00:00+00:00',
               '2012-07-06 14:30:00+00:00', '2012-07-06 15:30:00+00:00',
               '2012-07-06 16:30:00+00:00', '2012-07-06 17:30:00+00:00',
               '2012-07-06 18:30:00+00:00', '2012-07-06 19:30:00+00:00',
               '2012-07-06 20:00:00+00:00', '2012-07-09 14:30:00+00:00',
```

(continues on next page)

(continued from previous page)

```
                '2012-07-09 15:30:00+00:00', '2012-07-09 16:30:00+00:00',
                '2012-07-09 17:30:00+00:00', '2012-07-09 18:30:00+00:00',
                '2012-07-09 19:30:00+00:00', '2012-07-09 20:00:00+00:00',
                '2012-07-10 14:30:00+00:00', '2012-07-10 15:30:00+00:00',
                '2012-07-10 16:30:00+00:00', '2012-07-10 17:30:00+00:00',
                '2012-07-10 18:30:00+00:00', '2012-07-10 19:30:00+00:00',
                '2012-07-10 20:00:00+00:00'],
               dtype='datetime64[ns, UTC]', freq=None)
```

11.4.2 Merge schedules

```
# NYSE Calendar
nyse = mcal.get_calendar('NYSE')
schedule_nyse = nyse.schedule('2015-12-20', '2016-01-06')
schedule_nyse
```

```
# LSE Calendar
lse = mcal.get_calendar('LSE')
schedule_lse = lse.schedule('2015-12-20', '2016-01-06')
schedule_lse
```

Inner merge

This will find the dates where both the NYSE and LSE are open. Notice that Dec 28th is open for NYSE but not LSE. Also note that some days have a close prior to the open. This function does not currently check for that.

```
mcal.merge_schedules(schedules=[schedule_nyse, schedule_lse], how='inner')
```

Outer merge

This will return the dates and times where either the NYSE or the LSE are open

```
mcal.merge_schedules(schedules=[schedule_nyse, schedule_lse], how='outer')
```

11.4.3 Use holidays in numpy

This will use your exchange calendar in numpy to add business days

```
import numpy as np
cme = mcal.get_calendar("CME_Agriculture")
np.busday_offset(dates="2020-05-22", holidays=cme.holidays().holidays, offsets=1)
```

```
numpy.datetime64('2020-05-26')
```

11.4.4 Trading Breaks

Some markets have breaks in the day, like the CME Equity Futures markets which are closed from 4:15 - 4:35 (NY) daily. These calendars will have additional columns in the schedule() DataFrame

```
cme = mcal.get_calendar('CME_Equity')
schedule = cme.schedule('2020-01-01', '2020-01-04')
schedule
```

The date_range() properly accounts for the breaks

```
mcal.date_range(schedule, '5H')
```

```
DatetimeIndex(['2020-01-02 04:00:00+00:00', '2020-01-02 09:00:00+00:00',
               '2020-01-02 14:00:00+00:00', '2020-01-02 19:00:00+00:00',
               '2020-01-02 21:15:00+00:00', '2020-01-02 22:00:00+00:00',
               '2020-01-03 04:00:00+00:00', '2020-01-03 09:00:00+00:00',
               '2020-01-03 14:00:00+00:00', '2020-01-03 19:00:00+00:00',
               '2020-01-03 21:15:00+00:00', '2020-01-03 22:00:00+00:00'],
              dtype='datetime64[ns, UTC]', freq=None)
```

CHAPTER 12

New Market or Exchange

12.1 New Market or Exchange

See examples/usage.ipynb for demonstrations

To create a new exchange (or OTC market):

1. Create a new class that inherits from MarketCalendar
2. Set the class attribute *aliases: [...]* for accessing the calendar through *mcal.get_calendar*
3. Create the *regular_market_times* class attribute, meeting these requirements:
    1. It needs to be a dictionary
    2. Each market_time needs one entry
        1. Regular open must be “market_open”, regular close must be “market_close”.
        2. If there is a break, there must be a “break_start” and a “break_end”.
        3. only ONE break is currently supported.
    3. One tuple for each market_time, containing at least one tuple:
        1. Each nested tuple needs at least two items: *(first_date_used, time[, offset])*.
        2. The first tuple’s date should be None, marking the start. In every tuple thereafter this is the date when *time* was first used.
        3. Optionally (assumed to be zero, when not present), a positive or negative integer, representing an offset in number of days.
        4. Dates need to be in ascending order, None coming first.
4. Define the following property methods:
    1. name
    2. tz (time zone)

5. Now optionally define any of the following property methods:
    1. Days where the market is fully closed:
        1. regular_holidays - returns an pandas AbstractHolidayCalendar object
        2. adhoc_holidays - returns a list of pandas Timestamp of a DatetimeIndex
    2. Days where the market closes early:
        1. special_closes - returns a list of tuples. The tuple is (datetime.time of close, AbstractHolidayCalendar)
        2. special_closes_adhoc - returns a list of tuples. The tuple is (datetime.time of close, list of date strings)
    3. Days where the market opens late:
        1. special_opens - returns a list of tuples. The tuple is (datetime.time of open, AbstractHolidayCalendar)
        2. special_opens_adhoc - returns a list of tuples. The tuple is (datetime.time of open, list of date strings)
    4. Set special times for any market_time in regular_market_times, by setting a property in this format:
        1. **special_{market_time}_adhoc** same format as special_opens_adhoc, which is the same as special_market_open_adhoc
        2. **special_{market_time}** same format as special_opens, which is the same as special_market_open
    5. Add interruptions:
        1. interruptions - returns a list of tuples. The tuple is (date, start_time, end_time[, start_time2, end_time2, . . .])
6. Import your new calendar class in *calendar_registry.py*:

```
from .exchange_calendar_xxx import XXXExchangeCalendar
```

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

E

F

G

H

I

T

V

W