```
#!/usr/bin/python3.4
# -*-coding:Utf-8 -*


# The MIT License (MIT)
#
# Copyright (c) 2015-2016 Rémi Blaise <remi.blaise@gmx.fr> "http://php-zzortell.rhcloud.com/"
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.
#


import re


class EventDispatcher:
    '''
    A simple event dispatcher

    Author: Rémi Blaise (alias Zzortell) "http://php-zzortell.rhcloud.com/"

    '''


    def __init__(self, propagation=False):
        '''
        Init the event dispatcher

        Parameter:
        {bool} propagation = False If dispatching an event should also dispatch its parents

        '''

        self.propagation = propagation
        self.listeners = {}


    def listen(self, name, listener, priority=0):
        '''
        Add an event listener

        If name is 'all', the listener will listen all events.

        Parameters:
        {str}         name             The name of the event
        {function}     listener         The event listener
        {int}        priority = 0     The priority of the listener

        Return: {tuple} id The ID of the listener

        '''

        # Register listener
        if name not in self.listeners:
            self.listeners[name] = {}
        if priority not in self.listeners[name]:
            self.listeners[name][priority] = []
        self.listeners[name][priority].append(listener)

        return (name, priority, listener)


    def on(self, name):
        '''Inscribe given listener, to use as decorator '''
        def decorator(function):
            self.listen(name, function)
            return function
        return decorator


    def detach(self, id):
        '''
        Detach an event listener

        Parameter:
        {tuple} id The ID of the listener

        '''

        name, priority, listener  = id
        self.listeners[name][priority].remove(listener)


```

```
    def dispatch(self, name, event=None, propagation=None):
        '''
        Dispatch an event

        If propagation is set, dispatch all the parent events.

        Parameters:
        {str}         name                The name of the event
        {object}     event = None         The event to dispatch
        {bool}         propagation = None     Override self.propagation

        '''

        if name == 'all':
            raise ValueError("'all' is a reserved keyword, not an event name. ")
        propagation = propagation if propagation is not None else self.propagation

        # Get existing keys among ('all', name)
        names = []
        if 'all' in self.listeners:
            names.append('all')
        if name in self.listeners:
            names.append(name)

        # Get sorted list of priorities
        priorities = set()
        for name in names:
            priorities = priorities.union(set(self.listeners[name].keys()))
        priorities = list(priorities)
        priorities.sort()

        # Iterate over priorities
        for priority in priorities:
            # Get listeners
            listeners = []
            for name in names:
                if priority in self.listeners[name]:
                    listeners.extend(self.listeners[name][priority])

            # Iterate over listeners
            for listener in listeners:
                listener(event)

        # If propagation dispatch the parent event
        if propagation:
            parent_name = self.getParent(name)
            if parent_name:
                self.dispatch(parent_name, event)


    def getParent(self, name):
        '''
        Get the name of the parent event

        Used if the propagation option is True.
        The event name has to match the format "parent.event".

        Parameters:
        {str} name The name of the event

        Return: {str}     The name of the parent event
                None     If the event has no parent

        '''

        if re.search(r'^(?:\w+\.)*\w+$', name) is None:
            raise AssertionError("The event name has to match with r'^(?:\w+\.)*\w+$'. ")

        if re.search(r'\.', name):
            return re.search(r'^((?:\w+\.)*)\w+$', name).group(1)[:-1]
        else:
            return None
```