django$_{e}$*ventDocumentation*

Release 1.0.3

Dmitry Panchenko

June 24, 2015

Contents

Django Event is notification system framework that allows you to push notifications to the browser.

CHAPTER 1

Installing

First you need pip. Use your OS's package system to install it. On ubuntu you can install it like this:

```
apt-get install python-pip
```

And it is highly recommended you to use python virtual environment. See Virtualenv docs.

After that you need to install django-event:

```
pip install git+https://github.com/ailove-dev/django-event
```

If you use Django Rest Framework and want to django-event provided REST api's for events use following command:

```
pip install git+https://github.com/ailove-dev/django-event[rest_framework_api]
```

Contents:

1.1 django_event

1.1.1 django_event.management

django_event.management.commands

django_event.management.commands.deleteoldevents

class `django_event.management.commands.deleteoldevents.`**`Command`**

Bases: `django.core.management.base.NoArgsCommand`

Django command to delete old events.

`handle_noargs`(***options*)

django_event.management.commands.runwebsocketserver

class `django_event.management.commands.runwebsocketserver.`**`Command`**(*stdout=None*, *stderr=None*, *no_color=False*)

Bases: `django.core.management.base.BaseCommand`

Django command to start tornado server.

`handle`(**args*, ***options*)

1.1.2 django_event.publisher

django_event.publisher.rest_framework

django_event.publisher.rest_framework.serializers

Serializers module for django rest_framework.

class `django_event.publisher.rest_framework.serializers.`**`EventSerializer`**(*instance=None*, *data=None*, *files=None*, *context=None*, *partial=False*, *many=None*, *allow_add_remove=False*, ***kwargs*)

Bases: `rest_framework.serializers.ModelSerializer`

Specifies which fields should be serialized and overrides date fields to return local time instead of server time.

class `Meta`

`model`
alias of `Event`

`exclude` = ('event_request', 'task_name', 'task_id')

class `django_event.publisher.rest_framework.serializers.`**`LocalDateTimeField`**(*input_formats=None*, *format=None*, **args*, ***kwargs*)

Bases: `rest_framework.fields.DateTimeField`

Automatically converts server time into local time.

`to_representation`(*value*)
Converts UTC to local time.

Parameters **`value`** (`DateTime`) – UTC datetime from db.

django_event.publisher.rest_framework.urls

Urls for django.

```
urlpatterns = patterns(
    '',
    url(
        r'^$',
        event_list,
        name='event_list'
    ),
    url(
        r'^(?P<pk>[0-9]+)/$',
```

```
        event_detail,
        name='event_detail'
    ),
    url(
        r'^(?P<pk>[0-9]+)/cancel/$',
        cancel_event,
        name='cancel_event'
    ),
    url(
        r'^(?P<pk>[0-9]+)/retry/$',
        retry_event,
        name='retry_event'
    ),
)
```

django_event.publisher.rest_framework.views

Synchronous views for django to get/cancel or retry events.

class `django_event.publisher.rest_framework.views.`**`CancelEventView`**(***kwargs*)
Bases: `rest_framework.views.APIView`

Cancel executing event.

`permission_classes` **= (<class 'rest_framework.permissions.IsAuthenticated'>,)**

class `django_event.publisher.rest_framework.views.`**`EventDetailView`**(***kwargs*)
Bases: `rest_framework.views.APIView`

Retrieve specific user event.

`permission_classes` **= (<class 'rest_framework.permissions.IsAuthenticated'>,)**

`queryset`

`serializer_class`
alias of `EventSerializer`

class `django_event.publisher.rest_framework.views.`**`EventTypesView`**(***kwargs*)
Bases: `rest_framework.views.APIView`

Event types view.

`permission_classes` **= (<class 'rest_framework.permissions.IsAuthenticated'>,)**

class `django_event.publisher.rest_framework.views.`**`RetryEventView`**(***kwargs*)
Bases: `rest_framework.views.APIView`

Retry not completed event.

`permission_classes` **= (<class 'rest_framework.permissions.IsAuthenticated'>,)**

`django_event.publisher.rest_framework.views.`**`cancel_event`**(**args*, ***kwargs*)
Cancel executing event.

`django_event.publisher.rest_framework.views.`**`event_detail`**(**args*, ***kwargs*)
Retrieve specific user event.

`django_event.publisher.rest_framework.views.`**`event_list`**(**args*, ***kwargs*)
List all user events.

`django_event.publisher.rest_framework.views.`**`event_types`**(**args*, ***kwargs*)
Event types view.

`django_event.publisher.rest_framework.views.`**`retry_event`**(**args*, ***kwargs*)
Retry not completed event.

class `django_event.publisher.rest_framework.views.`**`EventListView`**(***kwargs*)
Bases: `rest_framework.generics.ListAPIView`

List all user events.

`filter_backends` **= (filters.OrderingFilter,)**
Filter backend. Provides basic ordering.

`get`(*request*, **args*, ***kwargs*)
Get user events —

`serializer_class`
alias of `EventSerializer`

django_event.publisher.decorator

Core publisher client module. Basically you should use this decorator instead of manually write events into database. Decorator specifies some overridable methods if you need basic customization.

Usage examples:

Defining event task:

```
@event(event_type='some_type', routing_strategy='')
def some_task(event_request, event):
    argument = event_request.custom_argument
    return some_processed_data(event_request.data, event)
```

And call:

```
some_task.delay(EventRequest(django_request, custom_argument=123))
```

You can know more about .delay() or other methods in Celery docs.

class `django_event.publisher.decorator.`**`event`**(*event_type=''*, *send_mail=False*, *progress_throttling=0.1*, *routing_strategy=''*, *task_kwargs=None*, *on_start=lambda _event: None*, *on_success=lambda _event: None*, *on_error=lambda _event: None*)
Bases: `object`

Main client publisher interface. Decorates function and return celery task.

Automatically start and end event after celery started/completed the task. You must pass EventRequest into decorated function. `EventRequest` can accept keyword arguments needed by wrapped function which are not required but django request are required by internal needs. This decorator automatically set `EventRequest` and `Event` instances as wrapped function arguments.

Wrapped func may raise specific exception `EventError` used to notify subscribers about failure. Basically this decorator will handle only that type of exceptions. Be sure you handle all raised exception in wrapped func otherwise it will pass outside decorator and into database as well.

`__call__`(*func*)
Wraps passed function into Celery `Task`.

Parameters **`func`** (*callable object*) – Function to be wrapped.

Returns Celery task instance.

Return type Celery `Task`

__init__(*event_type=''*, *send_mail=False*, *progress_throttling=0.1*, *routing_strategy=''*, *task_kwargs=None*, *on_start=lambda _event: None*, *on_success=lambda _event: None*, *on_error=lambda _event: None*)

Parameters

- **event_type** (`str`) – Event type, using in message routing.
- **send_mail** (`bool`) – Send email after event is done.
- **progress_throttling** (`float`) – Describes how often event will send progress messages. See `Event` docs for more information about this parameter.
- **routing_strategy** (`str`) – Routing strategy e.g. what listeners will do with messages. Empty strategy means you want to deliver notification to all subscribed clients. See `Event` docs for more information about this parameter.
- **task_kwargs** (`dict`) – Celery task arguments.
- **on_start** (`str`) – Event start callback.
- **on_success** (*callable object*) – Event success callback.
- **on_error** (*callable object*) – Event error callback.

complete_event()
Ends event. Override this if you want to customize message.

create_event()
Create event model with passing arguments. Basically you dont need to manually create event.

start_event()
Starts event. Override this if you want to customize message.

django_event.publisher.exceptions

Exceptions module.

exception `django_event.publisher.exceptions.`**EventError**
Bases: `exceptions.Exception`

Using to notify subscribed clients about event failure.

django_event.publisher.publisher

django_event.publisher.request

Event request module.

Usage examples: You can pass custom arguments into decorated function which are not required.

```
EventRequest(django_request, custom_argument=123)
```

But django request is must for event.

```
EventRequest(django_request)
```

class `django_event.publisher.request.`**`_DummyRequest`**(*data*, *user*)
Private class that emulates Django Request class.

class `django_event.publisher.request.`**`_DummyUser`**(*user_id*)
Private class that emulates Django User class.

class `django_event.publisher.request.`**`EventRequest`**(*django_request*, ***kwargs*)
Bases: `object`

Event request class user to pass arguments to decorated events.

`__getattr__`(*item*)
Magick method for dot notation for custom task arguments.

static `deserialize`(*json_request*, *return_dummy=False*)
Deserialize JSON request and returns EventRequest instance or dict which contains Dummy classes.

Parameters

- **`json_request`** (`str`) – JSON serialized event request.
- **`return_dummy`** (`bool`) – Return EventRequest or dict with dummies.

Returns Deserialized instance.

Return type *EventRequest* or `dict`

`serialize`()
Serialize event request into JSON for database storing.

Returns JSON serialized event request.

Return type `str`

django_event.publisher.views

Synchronous views for django to get/cancel or retry events.

class `django_event.publisher.views.`**`CancelEventView`**(***kwargs*)
Bases: *django_event.publisher.views.LoginRequiredMixin*, *django_event.publisher.views.EventDetailMixin*, `django.views.generic.base.View`

Base cancel event view class.

`cancel`(*pk*)
Returns True if event is successfully canceled False otherwise.

Parameters **`pk`** (`int`) – Event id.

Returns True if canceled else False.

Return type `bool`

`post`(*request*, *pk*)

class `django_event.publisher.views.`**`EventDetailMixin`**
Bases: *django_event.publisher.views.EventListMixin*

Event view mixin class with overrided *get_object()* method.

`get_object`(*pk*, *queryset=None*)
Returns user-owned event by id.

Returns QuerySet of user-owned events.

Return type `Event`

class django_event.publisher.views.**EventDetailView**(***kwargs*)
Bases: *django_event.publisher.views.LoginRequiredMixin*, *django_event.publisher.views.EventDetailMixin*, django.views.generic.detail.DetailView

Base event detail view class.

class django_event.publisher.views.**EventListMixin**
Bases: object

Event view mixin class with overrided *get_queryset()* method.

get_queryset()
Returns user-owned events.

Returns QuerySet of user-owned events.

Return type EventQuerySet

model
alias of Event

class django_event.publisher.views.**EventListView**(***kwargs*)
Bases: *django_event.publisher.views.LoginRequiredMixin*, *django_event.publisher.views.EventListMixin*, django.views.generic.list.ListView

Base event list view class.

class django_event.publisher.views.**EventTypesView**(***kwargs*)
Bases: *django_event.publisher.views.LoginRequiredMixin*, django.views.generic.base.View

Event types view.

get(*request*)
Return event types.

class django_event.publisher.views.**LoginRequiredMixin**
Bases: object

Event view mixin class with overrided *dispatch()* method to restrict anonymous users.

dispatch(**args*, ***kwargs*)
Override dispatch to implement auth restriction.

Parameters **request** (HttpRequest) – Incoming request.

Returns Response.

Return type HttpResponse

class django_event.publisher.views.**RetryEventView**(***kwargs*)
Bases: *django_event.publisher.views.LoginRequiredMixin*, *django_event.publisher.views.EventDetailMixin*, django.views.generic.base.View

Base retry event view class.

post(*request*, *pk*)

retry(*pk*)
Returns new event id if it successfully retried event None otherwise.

Parameters **pk** (int) – Event id.

Returns New event id if retried else None.

Return type int or None

django_event.publisher.views.**cancel_event**(*request*, **args*, ***kwargs*)
Base cancel event view class.

django_event.publisher.views.**event_detail**(*request*, **args*, ***kwargs*)
Base event detail view class.

django_event.publisher.views.**event_list**(*request*, **args*, ***kwargs*)
Base event list view class.

django_event.publisher.views.**event_types**(*request*, **args*, ***kwargs*)
Event types view.

django_event.publisher.views.**retry_event**(*request*, **args*, ***kwargs*)
Base retry event view class.

django_event.publisher.urls

Urls for django.

```
urlpatterns = patterns(
    '',
    url(
        r'^$',
        event_list,
        name='event_list'
    ),
    url(
        r'^(?P<pk>[0-9]+)/$',
        event_detail,
        name='event_detail'
    ),
    url(
        r'^(?P<pk>[0-9]+)/cancel/$',
        cancel_event,
        name='cancel_event'
    ),
    url(
        r'^(?P<pk>[0-9]+)/retry/$',
        retry_event,
        name='retry_event'
    ),
)
```

1.1.3 django_event.rabbitmq

django_event.rabbitmq.client

1.1.4 django_event.subscriber

django_event.subscriber.connection

Core event server module. Provides basic websocket connection and subscribe/unsubscribe message system.

class django_event.subscriber.connection.**_Request**
Bases: object

Private class that emulates Django Request class.

class `django_event.subscriber.connection.`**`EventConnection`**(*session*)
Bases: `sockjs.tornado.conn.SockJSConnection`

SockJS connection handler.

`executor`

Annotation = ThreadPoolExecutor(max_workers=cpu_count())

Thread pool for synchronous methods

`authenticate`(**args*, ***kwargs*)
Authenticates user by session key.

Parameters **`request`** (*Tornado Request*) – Request given on connection open.

Returns Authenticated user.

Return type Django User

`on_close`(**args*, ***kwargs*)
Asynchronous callback. Called internally by base class on connection closed. Unsubscribes all event listeners.

`on_message`(**args*, ***kwargs*)
Asynchronous callback. Called internally by base class on message received. Provides message routing. Subscribes and unsubscribes on messages types.

Parameters **`message`** (`str`) – JSON message received from RabbitMQ.

`on_open`(**args*, ***kwargs*)
Asynchronous callback. Basic authentication by session implemented. Called internally by base class on connection opened.

Parameters **`request`** (*Tornado Request*) – Request given on connection open.

`send`(**args*, ***kwargs*)
Sends message to websocket.

Parameters

- **`message`** (`dict`) – JSON decoded message routed by on_message method.
- **`binary`** (`bool`) – Flag. True if message need to be send in binary.

`subscribe`(**args*, ***kwargs*)
Subscribes event listeners on messages.

Parameters **`message`** (`dict`) – JSON decoded message routed by on_message method.

`unsubscribe`(**args*, ***kwargs*)
Unsubscribes event listeners from messages.

Parameters **`message`** (`dict`) – JSON decoded message routed by on_message method.

`wrong_message`(**args*, ***kwargs*)
Overridable method for wrong message type.

Parameters **`message`** (`dict`) – JSON decoded message routed by on_message method.

django_event.subscriber.listeners

Base listener module.

class `django_event.subscriber.listeners.`**`Listener`**(*user*)
Bases: `object`

Abstract base listener class. Listeners used by subscribers.

`executor` = **<concurrent.futures.thread.ThreadPoolExecutor object>**

classmethod **`get_listener`**(**args*, ***kwargs*)
Fabric method for listener subclasses. Imports listener by type and returns it.

Parameters **`event_type`** (`str`) – Event type.

Returns Listener module that will process certain type of messages.

Return type *`Listener`* subclass

`io_loop` = **<tornado.platform.epoll.EPollIOLoop object>**

`on_message`(*message*)
Abstract method. Specifies on message behaviour.

Parameters **`message`** (`str`) – Received message.

Raise NotImplementedError

`routing_matched`()
Computes routing key for current user and compares with received message's routing key.

Returns True if routing key matched.

Return type `bool`

class `django_event.subscriber.listeners.`**`SendMessageListener`**(*user*, *sender*)
Bases: *`django_event.subscriber.listeners.Listener`*

Base listener for send message through SockJS connection.

`on_message`(*message*)
Sends message if routing key matched.

Parameters **`message`** (`str`) – Received message.

django_event.subscriber.subscriber

1.1.5 django_event.utils

Utilities event module.

`django_event.utils.`**`get_routing`**(*user*, *routing_strategy*)
Get routing key needed by event or listener.

Parameters

- **`user`** (*User*) – User to be routed.
- **`routing_strategy`** (`str`) – How it can compute routing key.

Returns Routing key for RabbitMQ.

Return type `str`

`django_event.utils.`**`import_var`**(*var_path*)
Imports variable.

Parameters **`var_path`** (`str`) – Module path in python representation.

Returns Imported variable e.g. classes etc.

Return type Anything you import.

Raise `ImportError` if invalid module path passed into.

`django_event.utils.`**`try_import_or_runtime_error`**(*module*, *message*)

Trying to import module. No-op if imported. Raises `RuntimeError` with given message otherwise.

Parameters

- **`module`** (`str`) – Module name.
- **`message`** (`str`) – Error message.

Raise `RuntimeError` if module doesn't exist.

1.1.6 django_event.settings

Available django-event settings:

```
from django.conf import settings


event_settings = settings.DJANGO_EVENT


BACKEND = event_settings.get('BACKEND', 'rabbitmq')
BACKEND_OPTIONS = event_settings.get('BACKEND_OPTIONS', {})

TORNADO_OPTIONS = event_settings.get('TORNADO_OPTIONS', {})

LISTENERS = event_settings.get('LISTENERS', {})

STORE_DAYS = event_settings.get('STORE_DAYS', 7)

EVENT_MODEL = event_settings.get('EVENT_MODEL', 'django_event.Event')

AVAILABLE_TYPES = [key for key in LISTENERS.iterkeys()]
```

1.1.7 django_event.publisher.models

Core event model module.

You must handle old events by yourself using cron or celery worker.

class `django_event.models.`**`AbstractBaseEvent`**(**args*, ***kwargs*)

Bases: `django.db.models.base.Model`

An abstract base class implementing a fully featured Event model. Majority of model methods have custom_message parameter in case if you don't like/need default message protocol. Most of the methods are for private need i.e. all on_* methods. Be sure to not use these methods directly.

class `Meta`

`abstract` **= False**

`verbose_name` **= <django.utils.functional.__proxy__ object>**

`verbose_name_plural` **= <django.utils.functional.__proxy__ object>**

AbstractBaseEvent.**cancel**(*custom_message=None*)
Cancel executing event. If custom message passed it will send it instead of default message.

Parameters **custom_message** (*JSON serializable object.*) – Custom message.

AbstractBaseEvent.**complete**(*result*, *status=True*, *custom_message=None*, *callback=<function <lambda>>*, *errback=<function <lambda>>*)
Complete event and save state into database. It will always call passed callback.

Parameters

- **result** (*JSON serializable object.*) – Task result.
- **status** (bool) – Describes task result status e.g. task raised exception will set False status.
- **custom_message** (*JSON serializable object.*) – Custom message.
- **callback** (*callable object*) – Callback after event ended with success.
- **errback** (*callable object*) – Callback after event ended with error.

classmethod AbstractBaseEvent.**create**(*progress_throttling=0.1*, *routing_strategy=u''*, **args*, ***kwargs*)
Fabric method for event creature. Use this as main event creature source.

Parameters

- **progress_throttling** (float) – Describes how often event will send progress messages when user increments progress via *increment_progress()* method. *Event* will not send any progress messages with out user actions on *increment_progress()*. If you don't need progress you may not specify this parameter and not use *increment_progress()* method.
- **routing_strategy** (str) – Routing strategy e.g. what listeners will do with messages. Empty strategy means you want to deliver notification to all subscribed clients.
- **args** – Model init args
- **kwargs** – Model init kwargs

Returns Created and configured event.

Return type *Event*

classmethod AbstractBaseEvent.**delete_old**()
Deletes old events.

AbstractBaseEvent.**failure**
Shortcut for event status.

Returns True if event failed else False.

Return type bool

AbstractBaseEvent.**get_next_by_created_at**(**moreargs*, ***morekwargs*)

AbstractBaseEvent.**get_previous_by_created_at**(**moreargs*, ***morekwargs*)

AbstractBaseEvent.**increment_progress**(*progress_delta*, *custom_message=None*)
Increments event progress with out saving to database. If custom message passed it will send it instead of default message.

Parameters

- **progress_delta** (float) – Progress delta.

- **custom_message** (*JSON serializable object.*) – Custom message.

AbstractBaseEvent.**may_be_canceled**
Shortcut for event state.

Returns Check if event may be canceled.

Return type bool

AbstractBaseEvent.**may_be_retried**
Shortcut for event state.

Returns Check if event may be retried.

Return type bool

AbstractBaseEvent.**on_cancel**(*custom_message*)
Cancel callback. Sends message to subscribed clients.

Parameters **custom_message** (*JSON serializable object.*) – Custom message.

AbstractBaseEvent.**on_complete**(*custom_message*, *callback*, *errback*)
Complete callback. Dispatch between success and error status.

Parameters

- **custom_message** (*JSON serializable object.*) – Custom message.
- **callback** (*callable object*) – Callback after event ended with success.
- **errback** (*callable object*) – Callback after event ended with error.

AbstractBaseEvent.**on_error**(*custom_message*, *errback*)
Error callback. Sends message to subscribed clients.

Parameters

- **custom_message** (*JSON serializable object.*) – Custom message
- **errback** (*callable object*) – Errback after event ended with error.

AbstractBaseEvent.**on_progress_change**(*custom_message*)
Progress change callback. Sends message to subscribed clients.

Parameters **custom_message** (*JSON serializable object.*) – Custom message.

AbstractBaseEvent.**on_retry**(*custom_message*)
Cancel callback. Sends message to subscribed clients.

Parameters **custom_message** (*JSON serializable object.*) – Custom message.

AbstractBaseEvent.**on_start**(*custom_message*)
Start callback. Sends message to subscribed clients. If custom message passed it will send it instead of default message.

Parameters **custom_message** (*JSON serializable object.*) – Custom message.

AbstractBaseEvent.**on_success**(*custom_message*, *callback*)
Success callback. Sends message to subscribed clients.

Parameters

- **custom_message** (*JSON serializable object.*) – Custom message
- **callback** (*callable object*) – Callback after event ended with success.

AbstractBaseEvent.**retry**(*custom_message=None*, *request=None*, ***kwargs*)
Retry not completed event. If custom message passed it will send it instead of default message.

Parameters **custom_message** (*JSON serializable object.*) – Custom message.

AbstractBaseEvent.**send_email**()
Sends email to user after event is done.

TODO: implement email sending.

AbstractBaseEvent.**send_message**(*message*, *content_type*, *broadcast=False*, *callback=<function <lambda>>*)
Send custom message to subscribed clients and execute passed callback.

Parameters

- **message** (str) – Message.
- **content_type** (str) – Message content type.
- **broadcast** (bool) – True if you need broadcast.
- **callback** (*callable object*) – Callback after message send.

AbstractBaseEvent.**start**(*custom_message=None*, *callback=<function <lambda>>*)
Start event and save state into database. It will always call passed callback.

Parameters

- **custom_message** (*JSON serializable object.*) – Overrides default message.
- **callback** (*callable object*) – Callback after event started

AbstractBaseEvent.**success**
Shortcut for event status.

Returns True if event succeed else False.

Return type bool

AbstractBaseEvent.**try_custom**(*message*)
Custom message helper method. Publish message if present and return True, False otherwise.

Parameters **message** (*JSON serializable object.*) – Custom message.

AbstractBaseEvent.**user**

AbstractBaseEvent.**view**()
Mark instance as viewed.

class django_event.models.**EventQuerySet**(*model=None*, *query=None*, *using=None*, *hints=None*)
Bases: django.db.models.query.QuerySet

Event query set and manager. Defines some useful shortcuts.

completed()
Completed events.

Returns Completed events.

Return type *EventQuerySet*

failed()
Failed events.

Returns Failed events.

Return type *EventQuerySet*

mark_viewed()
Mark completed events as viewed.

not_completed()
Not completed events.

Returns Not completed events.

Return type *EventQuerySet*

not_viewed()
Not viewed events.

Returns Not viewed events.

Return type *EventQuerySet*

old()
Old events.

Returns Old events.

Return type *EventQuerySet*

successful()
Successful events.

Returns Successful events.

Return type *EventQuerySet*

viewed()
Viewed events.

Returns Viewed events.

Return type *EventQuerySet*

django_event.models.**get_event_model**()
Helper for getting event swappable model.

Returns *Event* model

class django_event.models.**Event**(**args*, ***kwargs*)
Default event model.

complete(*result*, *status=True*, *custom_message=None*, *callback=lambda event: None*, *errback=lambda event: None*)
Complete event and save state into database. It will always call passed callback.

Parameters

- **result** (*JSON serializable object.*) – Task result.
- **status** (bool) – Describes task result status e.g. task raised exception will set False status.
- **custom_message** (*JSON serializable object.*) – Custom message.
- **callback** (*callable object*) – Callback after event ended with success.
- **errback** (*callable object*) – Callback after event ended with error.

send_message(*message*, *content_type*, *broadcast=False*, *callback=lambda event: None*)
Send custom message to subscribed clients and execute passed callback.

Parameters

- **message** (`str`) – Message.
- **content_type** (`str`) – Message content type.
- **broadcast** (`bool`) – True if you need broadcast.
- **callback** (*callable object*) – Callback after message send.

start (*custom_message=None*, *callback=lambda event: None*)

Start event and save state into database. It will always call passed callback.

Parameters

- **custom_message** (*JSON serializable object.*) – Overrides default message.
- **callback** (*callable object*) – Callback after event started

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

d

Symbols

A

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W