Defining Custom Signals

All signals are django.dispatch.Signal instances.

class Signal():

Sending signals

There are two ways to send signals in Django.

- Signal.send(sender, **kwargs) – This is used to send a signal, all built-in signals use this to send signals. You must provide the sender argument which is a class most of the time and may provide as many other keyword arguments as you like. It returns a list of tuple pairs [(receiver, response), ...], representing the list of called receiver functions and their response values.
- Signal.send_robust(sender, **kwargs) – This is used to send a signal. You must provide the sender argument which is a class most of the time and may provide as many other keyword arguments as you like. It returns a list of tuple pairs [(receiver, response), ...], representing the list of called receiver functions and their response values.

Sending signals

Difference between send () and send_robust()

- send() does not catch any exceptions raised by receivers; it simply allows errors to propagate. Thus not all receivers may be notified of a signal in the face of an error.
- send_robust() catches all errors derived from Python's Exception class, and ensures all receivers are notified of the signal. If an error occurs, the error instance is returned in the tuple pair for the receiver that raised the error.

Disconnecting Signals

Signal.disconnect(receiver=None, sender=None, dispatch_uid=None) – This is used to disconnect a receiver from a signal. The arguments are as described in Signal.connect(). The method returns True if a receiver was disconnected and False if not.

The receiver argument indicates the registered receiver to disconnect. It may be None if dispatch_uid is used to identify the receiver.