

---

# rohrpost Documentation

**AX semantics**

**Aug 16, 2023**



---

## Contents:

---

<b>1</b>	<b>Protocol</b>	<b>3</b>
<b>2</b>	<b>Installation</b>	<b>5</b>
<b>3</b>	<b>Routing</b>	<b>7</b>
<b>4</b>	<b>Adding Handlers</b>	<b>9</b>
<b>5</b>	<b>Using the mixins</b>	<b>11</b>
<b>6</b>	<b>Utility functions</b>	<b>13</b>
<b>7</b>	<b>Migrating from rohrpost v2</b>	<b>15</b>
<b>8</b>	<b>Migrating from rohrpost v1</b>	<b>17</b>



Welcome to the rohrpost documentation!

rohrpost is a small library that aims to make protocol development for Django's sub-project [channels](#) (which provides WebSockets capabilities) easy and fun. We also have a client library called [rohrpost.js](#).

It features a light weight, JSON based protocol, including an exemplary handler implementing a ping/pong method. It also comes with a variety of helper functions, and Django model mixins that allow to automatically send updates when any user updates, deletes, or creates an object.



# CHAPTER 1

---

## Protocol

---

Therohrpost protocol sits on top of `channels`.rohrpost expects the messages to be valid JSON with

- An `id` field thatrohrpost sends back in the response.
- A `type` field that contains a string defining the message type (and hence, which handler will process the message).
- An optional `data` field containing whatever datarohrpost should pass to the handler. Please note thatrohrpost will pass all request data to the handler, and the naming of the `data` field is convention, not a rule.

A typical message would look like this:

```
{
  "id": 1234,
  "type": "ping",
  "data": [1, 2, 3, 4]
}
```

Both server and client messages look like this. When the server receives a message, it will hand off the message to the appropriate handler named in the `type` field. If there is no such handler or the handler fails, the server responds with a message containing at least an `error` field, and optionally other data in the `data` field.

```
{
  "id": 1234,
  "error": "No registered handler called 'ping'."
}
```



## CHAPTER 2

---

### Installation

---

From the command line:

```
pip install rohrpost
```

Or add this line to your *requirements.txt*:

```
rohrpost==3.*
```



Once you have installed *rohrpost*, you'll need to add a *rohrpost* consumer to your *routing.py*. You can find details on this in Channels' [routing documentation](#).

```
from django.urls import path
from rohrpost.sync_consumer import SyncRohrpostConsumer

websocket_urlpatterns = [
    path('ws/rohrpost/', SyncRohrpostConsumer),
]
```

If you use your own consumer implementation, make sure to a method `rohrpost_message` that handles messages sent over a channel layer to be broadcast to all members of the group. Further information is available in Channels documentation on [Channel layers](#).



---

## Adding Handlers

---

rohrpost provides a decorator `rohrpost_handler`, that accepts both a string and a list to register a method as the handler for incoming messages. This is how the `ping` method works, that rohrpost provides out of the box:

```
from rohrpost.message import send_message
from rohrpost.registry import rohrpost_handler

@rohrpost_handler('ping')
def handle_ping(consumer, request):
    send_message(
        consumer=consumer,
        message_id=request["id"],
        handler="pong",
        data=request.get("data"),
    )
```



---

## Using the mixins

---

We have four relevant Django model mixins in `rohrpost.mixins`: `NotifyOnCreate`, `NotifyOnUpdate`, `NotifyOnDelete` and `NotifyOnChange` which inherits from the previous three classes.

When using these mixins, you'll need to set some fields or fill in some methods:

- `get_group_name(self, message_type)` or `group_name`, with the method having preference over the attribute. This method or attribute should return a string that denotes the group receiving the message. All users in that group will receive a message. This gives you the possibility to build per-object, per-class or global groups. If neither the message nor the attribute are present, the group name is the lower cased class name combined with the object's ID: `f'{object.__class__.__name__.lower()}-{object.pk}'`
- `get_push_notification_data(self, updated_fields, message_type)` returning a dictionary (or any data structure) containing the data you wish to send to the client. `rohrpost` will update the serialized object updated with an `updated_fields` attribute with a list *if* you do not set `updated_fields` in `get_push_notification_data()` *and* if you set it when calling the `save()` method that lead to the notification. The fallback value if you do not fill in this method is `{"id": obj.id}`.

The message will look like this:

```
.. code-block:: JSON
```

```
{ "id": <some id>, "type": "subscription-update", "data": {
    "type": <createlupdateldelete>, "group": <group-name>, "object": <serialized object>,
  }
}
```

You will have to put users who should receive these notifications in the group specified by `get_group_name` or `group_name`. Since authentication and registration works differently in every use case, `rohrpost` does not include a standard handler for this.



---

## Utility functions

---

rohrpost provides a few helper functions for message sending in `rohrpost.message`:

- `rohrpost.message.send_message`
  - `consumer`: An instance of `channels.generic.websocket.WebsocketConsumer` that represents the client the message is sent to (**required**).
  - `handler`: The string identifying your handler (**required**).
  - `message_id`: The message ID (any simple datatype allowed). If you do not provide any, an integer will be randomly chosen.
  - `error`: Include an error message or error content
  - `data`: A dict that will appear in the message as `data` (converted to a JSON object).
- `rohrpost.message.send_error` sends an error message explicitly, takes the same arguments as `send_message`.
- `rohrpost.messages.send_to_group` sends a message to a specific group.



## CHAPTER 7

---

### Migrating from rohrpost v2

---

Make sure you are using Python 3.6+ and Channels 3+.



---

## Migrating from rohrpost v1

---

- Follow the [migration guide](#) from Channels 1 to 2.
- Adjust the routing in your application. What was before

```
channel_routing = [  
    route('websocket.receive', handle_rohrpost_message, path=r'^/ws/rohrpost/$'),  
]
```

should now be

```
websocket_urlpatterns = [  
    path('ws/rohrpost/', SyncRohrpostConsumer),  
]
```

- Your handlers will no longer receive `message` as a first argument, but an instance of `channels.generic.websocket.WebsocketConsumer`. You can pass it directly to the utility functions as before with `message`.
- To add clients to a group, change `Group(group_name).add(message.reply_channel)` to `consumer.add_to_group(group_name)` when using `rohrpost.sync_consumer.SyncRohrpostConsumer`. Otherwise we recommend to check out our consumer's implementation and read about [Groups in Channels 2](#).
- To send messages to a group you can use the new utility function `rohrpost.messages.send_to_group`.
- You can no longer pass `close` to the utility functions. Instead call `consumer.close()` after sending a message to close the connection.