Django Notification System

Release 1.0

Justin Branco, Michael Dunn

Dec 16, 2020

CONTENTS:

Perhaps you've got a Django application that you'd like to send notifications from?

Well, we certainly have our share of them. And guess what? We're tired of writing code to create and send various types of messages over and over again!

So, we've created this package to simplify things a bit for future projects. Hopefully, it will help you too.

Here's the stuff you get:

1. A few Django *models* that are pretty important:

- *Notification*: A single notification. Flexible enough to handle many different types of notifications.
- *NotificationTarget*: A target for notifications. Email, SMS, etc.
- *TargetUserRecord*: Info about the user in a given target (Ex. Your "address" in the "email" target).
- *NotificationOptOut*: Single location to keep track of user opt outs. You don't want the spam police after you.

2. Built in support for *email, Twilio SMS, and Expo push notifications.*.
3. Some cool management commands that:

- Process all pending notifications.
- Create *UserInNotificationTarget* objects for the email target for all the current users in your database. Just in case you are adding this to an older project.

4. A straightforward and fairly easy way to for you to add support for addition notification types while tying into the existing functionality. No whining about it not being super easy! This is still a work in progress. :)

Brought to you by the cool kids (er, kids that wanted to be cool) in the Center for Research Computing at Notre Dame.

CHAPTER

ONE

INSTALLATION

1.1 Requirements

- Python 3. Yes, we have completely ignored Python 2. Sad face.
- Django 3+
- A computer… preferrably plugged in.

1.2 Excuse me sir, may I have another?

Only the nerdiest of nerds put Dickens puns in their installation docs.

pip install django-notification-system

1.3 Post-Install Setup

Make the following additions to your Django settings.

Django Settings Additions

```
# You will need to add email information as specified here: https://docs.
↪djangoproject.com/en/3.1/topics/email/
# This can include:
EMAIL_HOST = ''
EMAIL_PORT = ''
EMAIL_HOST_USER = ''
EMAIL_HOST_PASSWORD = ''
# and the EMAIL_USE_TLS and EMAIL_USE_SSL settings control whether a secure␣
↪connection is used.

# Add the package to your installed apps.
INSTALLED_APPS = [
    "django_notification_system",
    ...
]

# Twilio Required settings, if you're not planning on using Twilio
# these can be set to empty strings
NOTIFICATION_SYSTEM_TARGETS={
  # Twilio Required settings, if you're not planning on using Twilio these can be␣
↪set
```

(continues on next page)

(continued from previous page)

```
  # to empty strings
  "twilio_sms": {
      'account_sid': '',
      'auth_token': '',
      'sender': '' # This is the phone number associated with the Twilio account
  },
  "email": {
      'from_email': '' # Sending email address
  }
}
```

If you would like to add support for addition types of notifications that don't exist in the package yet, you'll need to add some additional items to your Django settings. This is only necessary if you are planning on *extending the system*.

CHAPTER

TWO

PACKAGE MODELS

There are 4 models that the library will install in your application.

2.1 Notification Target

A notification target represents something that can receive a notication from our system. In this release of the package, we natively support Email, Twilio and Expo (push notifications) targets.

Unless you are *extending the system* you won't need to create any targets that are not already pre-loaded during installation.

2.1.1 Attributes

Key	Type	Description
id	uuid	Auto-generated record UUID.
name	str	The human friendly name for the target.
notification_module_name	str	The name of the module in the NOTIFICATION_SYSTEM_CREATORS & NOTIFICATION_SYSTEM_HANDLERS directories which will be used to create and process notifications for this target.

2.2 Target User Record

Each notification target will have an internal record for each of your users. For example, an email server would have a record of all the valid email addresses that it supports. This model is used to tie a Django user in your database to it's representation in a given *NotificationTarget*.

For example, for the built-in email target, we need to store the user's email address on a *TargetUserRecord* instance so that when we can the email *NotificationTarget* the correct address to send email notifications to for a given user.

2.2.1 Attributes

Key	Type	Description
id	uuid	Auto-generated record UUID.
user	Django User	The Django user instance associated with this record.
target	foreign key	The associated notification target instance.
target_user_id	str	The ID used in the target to uniquely identify the user.
description	str	A human friendly note about the user target.
active	boolean	Indicator of whether user target is active or not. For example, we may have an outdated email record for a user.

Example: Creating a Target User Record

```
from django.contrib.auth import get_user_model
from django_notification_system.models import (
    NotificationTarget, TargetUserRecord)

# Let's assume for our example here that your user model has a `phone_number`␣
↪attribute.
User = get_user_model()

user = User.objects.get(first_name="Eggs", last_name="Benedict")
target = NotificationTarget.objects.get(name='Twilio')

# Create a target user record.
target_user_record = TargetUserRecord.objects.create(
    user=user,
    target=target,
    target_user_id=user.phone_number,
    description=f"{user.first_name} {user.last_name}'s Twilio",
    active=True
)
```

2.3 Notification Opt Out

Use this model to track whether or not users have opted-out of receiving notifications from you.

- For the built in *Process Notifications* command, we ensure that notifications are not sent to users with active opt-outs.
- Make sure to check this yourself if you implement other ways of sending notifications or you may find yourself running afoul of spam rules.

2.3.1 Attributes

Key	Type	Description
user	Django User	The Django user associated with this record.
active	boolean	Indicator for whether the opt out is active or not.

Example: Creating an Opt out

```
from django.contrib.auth import get_user_model
from django_notification_system.models import NotificationOptOut

User = get_user_model()
user = User.objects.get(first_name="Eggs", last_name="Benedict")

opt_out = NotificationOptOut.objects.create(
    user=user,
    active=True)
```

2.3.2 Unique Behavior

When an instance of this model is saved, if the opt out is *active* existing notifications with a current status of SCHEDULED or RETRY will be changed to OPTED_OUT.

We do this to help prevent them from being sent, but also to keep a record of what notifications had been scheduled before the user opted-out.

2.4 Notification

This model represents a notification in the database. SHOCKING!

Thus far, we've found this model to be flexible enough to handle any type of notification. Hopefully, you will find the same.

2.4.1 Core Concept

Each type of notification target must have a corresponding handler module that will process notifications that belong to that target. These handlers interpret the various attributes of a *Notification* instance to construct a valid message for each target.

For each of the built-in targets, we have already written these handlers. If you create additional targets, you'll need to write the corresponding handlers. See the *extending the system* page for more information.

2.4.2 Attributes

Key	Type	Description
target_user_record	TargetUserRecord	The TargetUserRecord associated with notification. This essentially identifies the both the target (i.e. email) and the specific user in that target (coolkid@nd.edu) that will receive the notification.
title	str	The title for the notification.
body	str	The main message of the notification to be sent.
extra	dict	A dictionary of extra data to be sent to the notification handler. Valid keys are determined by each handler.
status	str	The status of Notification. Options are: 'SCHEDULED', 'DELIVERED', 'DELIVERY FAILURE', 'RETRY', 'INACTIVE DEVICE', 'OPTED OUT'
scheduled_delivery	DateTime	Scheduled delivery date/time.
attempted_delivery	DateTime	Last attempted delivery date/time.
retry_time_interval	PositiveInt	If a notification delivery fails, this is the amount of time to wait until retrying to send it.
retry_attempts	PositiveInt	The number of delivery retries that have been attempted.
max_retries	PositiveInt	The maximun number of allowed delivery attempts.

Example: Creating an Email Notification

```
from django.contrib.auth import get_user_model
from django.utils import timezone

from django_notification_system.models import UserInNotificationTarget,␣
↪Notification

# Get the user.
User = get_user_model()
user = User.objects.get(first_name="Eggs", last_name="Benedict")

# The the user's target record for the email target.
emailUserRecord = TargetUserRecord.objects.get(
    user=User,
    target__name='Email')

# Create the notification instance.
# IMPORTANT: This does NOT send the notification, just schedules it.
# See the docs on management commands for sending notifications.
notification = Notification.objects.create(
        user_target=user_target,
        title=f"Good morning, {user.first_name}",
        body="lorem ipsum...",
        status="SCHEDULED",
        scheduled_delivery=timezone.now()
)
```

2.4.3 Unique Behavior

We perform a few data checks whenever an notification instance is saved.

1. You cannot set the status of notification to 'SCHEDULED' if you also have an existing attempted delivery date.
2. If a notification has a status other than 'SCHEDULED' or 'OPTED OUT it MUST have an attempted delivery date.
3. Don't allow notifications to be saved if the user has opted out.

CHAPTER

THREE

MANAGEMENT COMMANDS

Alright friends, in additional to all the goodies we've already talked about, we've got a couple of management commands to make your life easier. Like, a lot easier.

3.1 Process Notifications

This is the big kahuna of the entire system. When run, this command will attempt to deliver all notifications with a status of *SCHEDULED* or *RETRY* whose `scheduled_delivery` attribute is anytime before the command was invoked.

3.1.1 How to Run it

```
$ python manage.py process_notifications
```

Make Life Easy for Yourself

Once you've ironed out any potential kinks in your system, consider setting up a CRON schedule for this command that runs at an appropriate interval for your application. After that, your notifications will fly off your database shelves to your users without any further work on your end.

3.1.2 Important: If You Have Custom Notification Targets

If you have created custom notification targets, you MUST have created the appropriate handler modules. You can find about how to do this *here.*

If this isn't done, no notifications for custom targets will be sent.

3.1.3 Example Usage

Creating Notifications

```
# First, we'll need to have some Notifications in our database
# in order for this command to send anything.
from django.contrib.auth import get_user_model
from django.utils import timezone

from django_notification_system.models import (
```

(continues on next page)

(continued from previous page)

```
    TargetUserRecord, Notification)

User = get_user_model()

user = User.objects.get(first_name="Eggs", last_name="Benedict")

# Let's assume this user has 3 TargetUserRecord objects,
# one for Expo, one for Twilio and one for Email.
user_targets = TargetUserRecord.objects.filter(
    user=user)

# We'll loop through these targets and create a basic notification
# instance for each one.
for user_target in user_targets:
    Notification.objects.create(
        user_target=user_target,
        title=f"Test notification for {user.first_name} {user.last_name}",
        body="lorem ipsum...",
        status="SCHEDULED,
        scheduled_delivery=timezone.now()
    )
```

Now we have three Notifications ready to send. Let's run the command.

```
$ python manage.py process_notifications
```

If all was successful, you will see the output below. What this means is that all Notifications (1) were sent and (2) have been updated to have a `status` of 'DELIVERED' and an `attempted_delivery` set to the time it was sent.

```
egg - 2020-12-06 19:57:38+00:00 - SCHEDULED
Test notification for Eggs Benedict - lorem ipsum...
SMS Successfully sent!
******************************
egg - 2020-12-06 19:57:38+00:00 - SCHEDULED
Test notification for Eggs Benedict - lorem ipsum...
Email Successfully Sent
******************************
egg - 2020-12-06 19:57:38+00:00 - SCHEDULED
Test notification for Eggs Benedict - lorem ipsum...
Notification Successfully Pushed!
******************************
```

If any error occurs, that will be captured in the output. Based on the `retry` attribute, the affected notification(s) will try sending the next time the command is invoked.

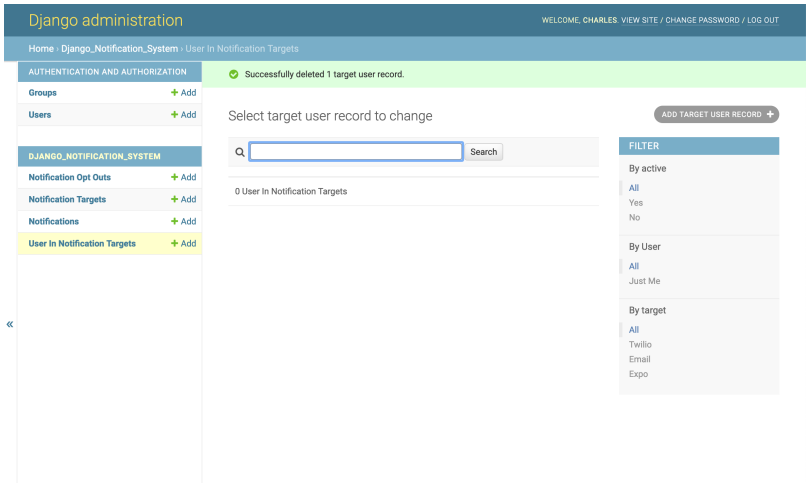
3.2 Create Email Target User Records

The purpose of this command is to create an email target user record for each user currently in your database or update them if they already exist. We do this by inspecting the `email` attribute of the user object and creating/updating the corresponding notification system models as needed.

After initial installation of this package, we can see that the `User Targets` section of our admin panel is empty.

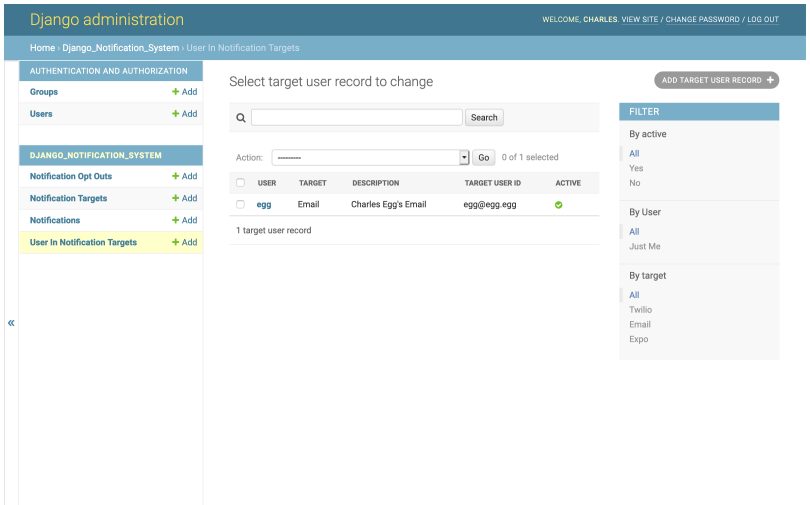
Oh no!

FEAR NOT! In your terminal, run the command:




```
$ python manage.py create_email_target_user_records
```

After the command has been run, navigate to `http://yoursite/admin/django_notification_system/targetuserrecord/`. You should see a newly created UserInNotificationTarget for each user currently in the DB.



These user targets are now available for all of your notification needs.

CHAPTER

FOUR

BUILT-IN NOTIFICATION CREATORS & HANDLERS

What allows for a given notification type to be supported is the existence of a **notification creator** and **notification handler** functions. Their jobs are to:

1. Create a `Notification` record for a given notification target.
2. Interpret a `Notification` record in an appropriate way for a given target and actually send the notification.

Currently there are 3 different types of notifications with built-in support:

- Email
- Twilio SMS
- Expo Push

4.1 Natively Supported Notification Targets

4.1.1 Email Notifications

NOTE: To send emails, you will need to have the appropriate variables in your settings file. More information can be found . We also have examples *here.*

Notification Creator

Example: Email Notification Creator

```
from django.contrib.auth import get_user_model

from django_notification_system.notification_creators.email import create_
↪notification

User = get_user_model()

user = User.objects.get(first_name="Eggs", last_name="Benedict")

# Note how the extra parameter is used here.
# See function parameters below for more details.
create_notification(
    user=user,
    title='Cool Email',
    extra={
        "user": user,
```

(continues on next page)

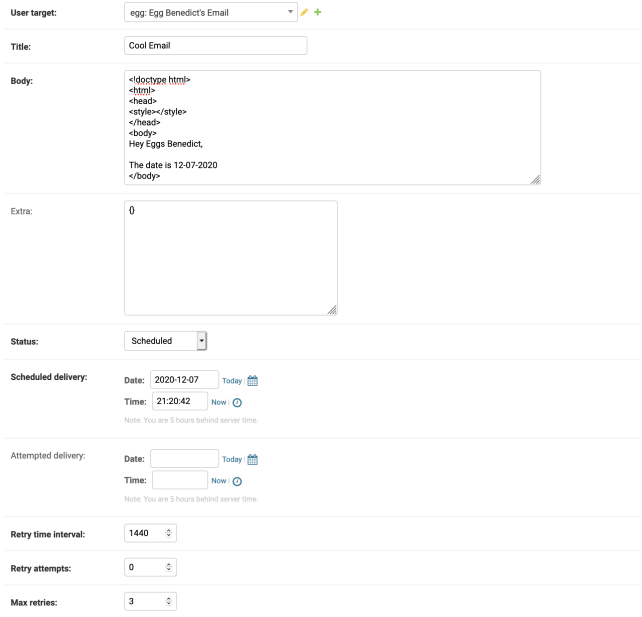
(continued from previous page)

```
        "date": "12-07-2020"
        "template_name": "templates/eggs_email.html"
    })
```

Function Parameters

Key	Type	Description
user	Django User	The user to whom the notification will be sent.
title	str	The title for the notification.
body	str	Body of the email. Defaults to a blank string if not given. Additionally, if this parameter is not specific AND "template_name" is present in *extra*, an attempt will be made to generate the body from that template.
scheduled_delivery	datetime(optional)	When to delivery the notification. Defaults to immediately.
retry_time_interval	int(optional)	When to retry sending the notification if a delivery failure occurs. Defaults to 1440 seconds.
max_retries	int(optional)	Maximum number of retry attempts. Defaults to 3.
quiet	bool(optional)	Suppress exceptions from being raised. Defaults to False.
extra	dict(optional)	User specified additional data that will be used to populate an HTML template if "template_name" is present inside.

The above example will create a Notification with the following values:



Notification Handler

Example Usage

```
from django.utils import timezone

from django_notification_system.models import Notification
from django_notification_system.notification_handlers.email import send_
↪notification

# Get all email notifications.
notifications_to_send = Notification.objects.filter(
    target_user_record__target__name='Email',
    status='SCHEDULED',
    scheduled_delivery__lte=timezone.now())

# Send each email notification to the handler.
for notification in notifications_to_send:
    send_notification(notification)
```

4.1.2 Expo Push Notifications

Notification Creator

Example: Expo Notification Creator

```
from django.contrib.auth import get_user_model

from django_notification_system.notification_creators.expo import create_
↪notification

User = get_user_model()

user = User.objects.get(first_name="Eggs", last_name="Benedict")

create_notification(
    user=user,
    title=f"Hello {user.first_name}",
    body="Test push notification")
```

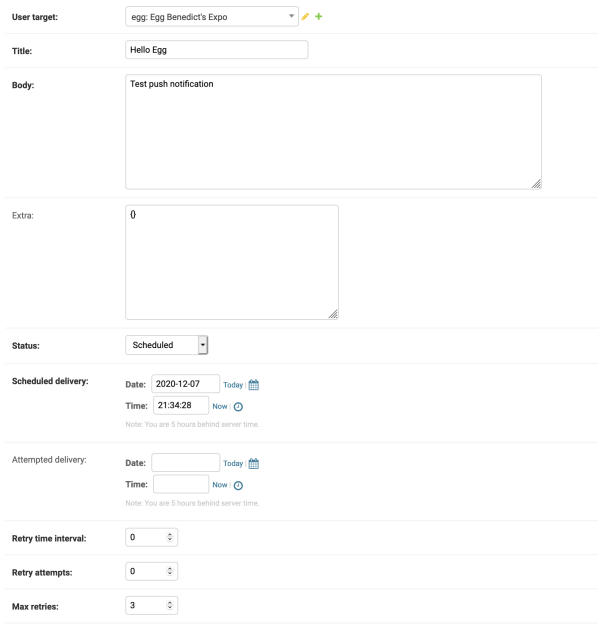
Parameters

Key	Type	Description
user	Django User	The user to whom the notification will be sent.
title	str	The title for the push notification.
body	str	The body of the push notification.
scheduled_delivery	datetime(optional)	When to delivery the notification. Defaults to immediately.
retry_time_interval	int(optional)	Delay between send attempts. Defaults to 60 seconds.
max_retries	int(optional)	Maximum number of retry attempts. Defaults to 3.
quiet	bool(optional)	Suppress exceptions from being raised. Defaults to False.
extra	dict(optional)	Defaults to None.

The above example will create a Notification with the following values:

Notification Handler

Example Usage




```
from django.utils import timezone

from django_notification_system.models import Notification
from django_notification_system.notification_handlers.expo import send_
↪notification

# Get all Expo notifications.
notifications_to_send = Notification.objects.filter(
    target_user_record__target__name='Expo',
    status='SCHEDULED',
    scheduled_delivery__lte=timezone.now())

# Send each Expo notification to the handler.
for notification in notifications_to_send:
    send_notification(notification)
```

4.1.3 Twilio SMS

NOTE: All Twilio phone numbers must contain a + and the country code. Therefore, all Twilio UserTargetRecords target_user_id should be '+{country_code}7891234567'. The sender number stored in the settings file should also follow this format.

Notification Creator

Example: Twilio SMS Notification Creator

```
from django.contrib.auth import get_user_model

from django_notification_system.notification_creators.twilio import create_
↪notification
```

(continues on next page)

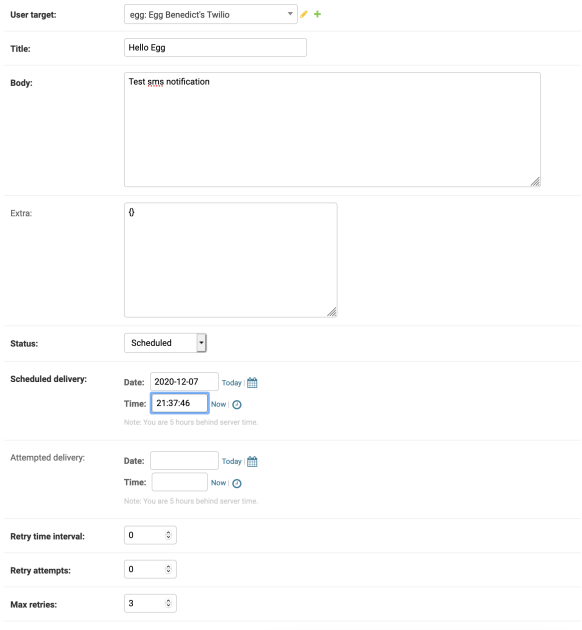
(continued from previous page)

```
User = get_user_model()

user = User.objects.get(first_name="Eggs", last_name="Benedict")

create_notification(
    user=user,
    title=f"Hello {user.first_name}",
    body="Test sms notification")
```

Parameters

Key	Type	Description
user	Django User	The user to whom the notification will be sent.
title	str	The title for the sms notification.
body	str	The body of the sms notification.
scheduled_delivery	datetime(optional)	When to deliver the notification. Defaults to immediately.
retry_time_interval	int(optional)	Delay between send attempts. Defaults to 60 seconds.
max_retries	int(optional)	Maximum number of retry attempts. Defaults to 3.
quiet	bool(optional)	Suppress exceptions from being raised. Defaults to False.
extra	dict(optional)	Defaults to None.

The above example will create a Notification with the following values:



Notification Handler

Example Usage

```
from django.utils import timezone

from django_notification_system.models import Notification
```

(continues on next page)

(continued from previous page)

```
from django_notification_system.notification_handlers.twilio import send_
↪notification

# Get all notifications for Twilio target.
notifications_to_send = Notification.objects.filter(
    target_user_record__target__name='Twilio',
    status='SCHEDULED',
    scheduled_delivery__lte=timezone.now())

# Send each notification to the Twilio handler.
for notification in notifications_to_send:
    send_notification(notification)
```

CHAPTER

FIVE

ADDING SUPPORT FOR CUSTOM NOTIFICATION TARGETS

5.1 Option 1: Beg us to do it.

In terms of easy to do, this would be at the top of the list. However, we've got to be honest. We're crazy busy usually, so the chances that we will be able to do this aren't great. However, if we see a request that we think would have a lot of mileage in it we may take it up.

If you want to try this method, just submit an issue on the

5.2 Option 2: Add Support Yourself

Ok, you can do this! It's actually pretty easy. Here is the big picture. Let's go through it step by step.

5.2.1 Step 1: Add Required Django Settings

The first step is to tell Django where to look for custom notification creators and handlers. Here is how you do that.

Django Settings Additions

```
# A list of locations for the system to search for notification creators.
# For each location listed, each module will be searched for a `create_
↪notification` function.
NOTIFICATION_SYSTEM_CREATORS = [
    '/path/to/creator_modules',
    '/another/path/to/creator_modules']

# A list of locations for the system to search for notification handlers.
# For each location listed, each module will be searched for a `send_
↪notification` function.
NOTIFICATION_SYSTEM_HANDLERS = [
    '/path/to/handler_modules',
    '/another/path/to/handler_modules']
```

5.2.2 Step 2: Create the Notification Target

Now that you've added the required Django settings, we need to create a `NotificationTarget` object for your custom target.

Example: Creating a New Notification Target

```
from django_notification_system.models import NotificationTarget

# Note: The notification_module_name will be the name of the modules you will␣
↪write
# to support the new notification target.
# Example: If in my settings I have the NOTIFICATION_SYSTEM_HANDLERS = ["/path/to/
↪extra_handlers"],
# and inside that directory I have a file called 'carrier_pigeon.py', the␣
↪notification_module_name should be 'carrier_pigeon'
target = NotificationTarget.objects.create(
    name='Carrier Pigeon',
    notification_module_name='carrier_pigeon')
```

5.2.3 Step 2: Add a Notification Creator

Next, we need to create the corresponding creator and handler functions. We'll start with the handler function.

In the example above, you created a `NotificationTarget` and set it's `notification_module_name` to `carrier_pigeon`. This means that the `process_notifications` management command is going to look for modules named `carrier_pigeon` in the paths specified by your Django settings additions to find the necessary creator and handler functions.

Let's start by writing our creator function.

Example: Creating the Carrier Pigeon Notification Creator

```
# /path/to/creators/carrier_pigeon.py

from datetime import datetime
from django.utils import timezone
from django.contrib.auth import get_user_model

# Some common exceptions you might want to use.
from django_notification_system.exceptions import (
    NotificationsNotCreated,
    UserHasNoTargetRecords,
    UserIsOptedOut,
)

# A utility function to see if the user has an opt-out.
from django_notification_system.utils import (
    check_for_user_opt_out
)

from ..models import Notification, TargetUserRecord

# NOTE: The function MUST be named `create_notification`
def create_notification(
    user: 'Django User',
    title: str,
    body: str,
    scheduled_delivery: datetime = None,
    retry_time_interval: int = 60,
    max_retries: int = 3,
    quiet=False,
    extra: dict = None,
```

(continues on next page)

(continued from previous page)

```
) -> None:
    """
    Create a Carrier Pigeon notification.

    Args:
        user (User): The user to whom the notification will be sent.
        title (str): The title for the notification.
        body (str): The body of the notification.
        scheduled_delivery (datetime, optional): Defaults to immediately.
        retry_time_interval (int, optional): Delay between send attempts.␣
↪Defaults to 60 seconds.
        max_retries (int, optional): Maximum number of retry attempts for␣
↪delivery. Defaults to 3.
        quiet (bool, optional): Suppress exceptions from being raised. Defaults␣
↪to False.
        extra (dict, optional): Defaults to None.

    Raises:
        UserIsOptedOut: When the user has an active opt-out.
        UserHasNoTargetRecords: When the user has no eligible targets for this␣
↪notification type.
        NotificationsNotCreated: When the notifications could not be created.
    """

    # Check if user is opted-out.
    try:
        check_for_user_opt_out(user=user)
    except UserIsOptedOut:
        if quiet:
            return
        else:
            raise UserIsOptedOut()

    # Grab all active TargetUserRecords in the Carrier Pigeon target
    # the user has. You NEVER KNOW if they might have more than one pigeon.
    carrier_pigeon_user_records = TargetUserRecord.objects.filter(
        user=user,
        target__name="Carrier Pigeon",
        active=True,
    )

    # If the user has no active carrier pigions, we
    # can't create any notifications for them.
    if not carrier_pigeon_user_records:
        if quiet:
            return
        else:
            raise UserHasNoTargetRecords()

    # Provide a default scheduled delivery if none is provided.
    if scheduled_delivery is None:
            scheduled_delivery = timezone.now()

    notifications_created = []
    for record in carrier_pigeon_user_records:

        if extra is None:
```

(continues on next page)

(continued from previous page)

```
            extra = {}

        # Create notifications while taking some precautions
        # not to duplicate ones that are already there.
        notification, created = Notification.objects.get_or_create(
            target_user_record=record,
            title=title,
            scheduled_delivery=scheduled_delivery,
            extra=extra,
            defaults={
                    "body": body,
                    "status": "SCHEDULED",
                    "retry_time_interval": retry_time_interval,
                    "max_retries": max_retries,
                },
            )

        # If a new notification was created, add it to the list.
        if created:
            notifications_created.append(notification)

    # If no notifications were created, possibly raise an exception.
    if not notifications_created:
        if quiet:
            return
        else:
            raise NotificationsNotCreated()
```

5.2.4 Step 3: Add a Notification Handler

Alright my friend, last step. The final thing you need to do is write a notification handler. These are used by the *process_notifications* management command to actual send the notifications to the various targets.

For the sake of illustration, we'll continue with our carrier pigeon example.

Example: Creating the Carrier Pigeon Notification Handler

```
# /path/to/hanlders/carrier_pigeon.py

from dateutil.relativedelta import relativedelta
from django.utils import timezone

# Usually, the notification provider will have either an
# existing Python SDK or RestFUL API which your handler
# will need to interact with.
from carrior_pigeon_sdk import (
    request_delivery,
    request_priority_delivery,
    request_economy_aka_old_pigeon_delivery
    PigeonDiedException,
    PigeonGotLostException
)

from ..utils import check_and_update_retry_attempts

# You MUST have a function called send_notification in this module.
```

(continues on next page)

(continued from previous page)

```
def send_notification(notification) -> str:
    """
    Send a notification to the carrior pigeon service for delivery.

    Args:
        notification (Notification): The notification to be delivery by carrior␣
↪pigeon.

    Returns:
        str: Whether the push notification has successfully sent, or an error␣
↪message.
    """
    try:
        # Invoke whatever method of the target service you need to.
        # Notice how the handler is responsible to translate data
        # from the `Notification` record to what is needed by the service.
        response = request_delivery(
            recipient=notification.target_user_record.target_user_id,
            sender="My Cool App",
            title=notification.title,
            body=notification.body,
            talking_pigeon=True if "speak_message" in test and extra["speak_
↪message"] else False,
            pay_on_delivery=True if "cheapskate" in test and extra["cheapskate"]␣
↪else False
        )

    except PigeonDiedException as error:
        # Probably not going to be able to reattempt delivery.
        notification.attempted_delivery = timezone.now()
        notification.status = notification.DELIVERY_FAILURE
        notification.save()

        # This string will be displayed by the
        # `process_notifications` management command.
        return "Yeah, so, your pigeon died. Wah wah."

    except PigeonGotLostException as error:
        notification.attempted_delivery = timezone.now()

        # In this case, it is possible to attempt another delivery.
        # BUT, we should check if the max attempts have been made.
        if notification.retry_attempts < notification.max_retries:
            notification.status = notification.RETRY
            notification.scheduled_delivery = timezone.now() + relativedelta(
                minutes=notification.retry_time_interval)
            notification.save()
            return "Your bird got lost, but we'll give it another try later."
        else:
            notification.status = notification.DELIVERY_FAILURE
            notification.save()
            return "Your bird got really dumb and keeps getting lost. And it ate␣
↪your message."
```

5.3 Option 3: Be a cool kid superstar.

Write your own custom stuff and submit a PR to share with others.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search