EasyNewsletter Documentation

Release 2.5.6

Kai Diefenbach and the Plone Community

Feb 18, 2018

Contents

CHAPTER 1

What is it?

EasyNewsletter is a simple but powerful newsletter/mailing product for Plone.

For Features, Requirements and Installation see README on PyPi.

CHAPTER 2

Usage

2.1 General

You can use EasyNewsletter to manually create mailings/newsletters or you can use the collection criteria to collect content.

EasyNewsletter can use multible Collections to aggregate content for a newsletter issue.

Once the content is generated one can edit the text as usual in Plone. But it's not recommented, because TinyMCE is not very helpful with email markup.

You can create your own templates to structure the selected content. Please refer to the provided templates to see how it works.

2.2 Step by step

1. Add a EasyNewsletter instance, fill in the form and save it.
2. If you want to write a simple manual mailing, you can add an Issue and fill it out with your text.
3. Or if you want to use collections to aggregate your content first, select some existing Collection in `Content aggregation sources` on the Newsletter and finally add an Issue.
4. You can select more than one Collection to build categories like news, events and pictures in your newsletter. Empty Collections will be ignored in default aggregations templates.
5. Go to the view tab and call `Aggregate body content` from the action or Newsletter menu (>= Plone 5) menu.
6. You can also select *Content aggregation sources'* on Issue level. Then only the issue Collections are used.
7. Go to Send tab or Send action in Newsletter menu (>=Plone 5) and push `Test Newsletter`.
8. If your Newsletter/Mailing is finished, you can activate the send button by clicking on `Enable send button`. Then you can click on `Send newsletter` to send the Newsletter to all subscribers or selected groups and users.

2.3 Issue workflow information

There are now four workflow states for an issue, which are draft, sending, sent and master. If an issue is created, it's initial state is draft.

Only issues with state draft can be send.

If a sendout is started, the state will move to sending.

After an issue is sent, it's state is sent and it will appear in the newsletter archive.

In addition a master can be made out of an issue with the state draft or sent using the actions menu. The master acts as a blueprint, which can be reedited and copied as a new draft.

2.4 Images in HTML mails

- All images with local urls in `src` attribute, are included and attached to the mail.
- All images with external absolute urls in `src` attribute are not attached but included in HTML with the original `src` url.

2.5 Elements for mails only

If you want some elements, let's say a logo only in mails but not in the public view, you can add a class `mailonly`. All elements with class `mailonly` are filtered out in the public view.

2.6 Asyncronous sendout

Products.EasyNewsletter supports asyncronous sendout using collective.taskqueue

Add this to your instance section in your buildout config:

```
zope-conf-additional =
    <taskqueue>
      queue Products.EasyNewsletter.queue
    </taskqueue>
    <taskqueue-server>
      queue Products.EasyNewsletter.queue
    </taskqueue-server>
```

In your eggs list you should add the following install extra `[taskqueue]`:

```
Products.EasyNewsletter[taskqueue]
```

This will install collective.taskqeue as requirement. If you have configured your buildout according accordingly, Products.EasyNewsletter will automatically delegate the sendout to your worker instance.

2.7 Sending a daily issue automatically

EasyNewsletter can create and send daily issues, using the default template and default criteria. Beside those, you just need to configure your crontab (or clock server) to send a *POST* on *@@daily-issue* view. Eg:

```
#Sends a newsletter, from Mon to Fri, at 0:00AM
0 0 * * 1-5 curl -X POST http://user:passwd@example.org/mynewsletter/@@daily-issue
```

@@daily-issue returns a HTTP status code indicating what just happened (you can also test it with a GET, instead). In the table below, you can check the responses codes.

@@daily-issue responses			
Method/Precondition	Not yet sent	Empty	Already Sent
GET	100	204	200
POST	200*[0]	204	409

2.7.1 Using cron4plone to send it out

You can now use cron4plone to call the new trigger `@@trigger-daily-issue` which will make a POST request to the `@@daily-issue`.

2.8 Filtering Users, Groups and Receivers

EasyNewsletter provide a flexible way to filter the Plone members, Plone groups and the recivers list. You can provide smal funtion in your add-on and register it as IReceiversMemberFilter, IReceiversGroupFilter or IReceiversPostSendingFilter. The filter get the list and can filter out some entries or even add some entries.

`Interface: Products.EasyNewsletter.interfaces.IReceiversMemberFilter`

The IReceiversMemberFilter filters can be used to filter the list of Plone members which a user can select in newsletters and issues.

```
class ReceiversMemberFilterNoPloneMember(object):
    """ filters all members out of newsletter receivers selection list,
        which are default plone members. This is usefull if you whant
        only membrane members but not the default plone user as receivers.
        receivers: [(id, {'email': 'info@example.com',...})]
    """

    def __init__(self, context):
        self.context = context

    def filter(self, receivers):
        portal = getSite()
        query = {}
        query['portal_type'] = ['Contact',]
        contacts = portal.membrane_tool.search(query)
        whitelist = [contact.getUserId for contact in contacts]
        receivers = [receiver for receiver in receivers
                if receiver[0] in whitelist]
        return receivers
```

This filter should be registered as follow:

[0] It sends the issue first.

```
<subscriber zcml:condition="installed my.package"
    for="Products.EasyNewsletter.interfaces.IEasyNewsletter"
    factory="my.package.newsletter.ReceiversMemberFilterNoPloneMember"
    provides="Products.EasyNewsletter.interfaces.IReceiversMemberFilter" />
```

`Interface: Products.EasyNewsletter.interfaces.IReceiversGroupFilter`

The IReceiversGroupFilter filters can be used to filter the list of Plone groups which a user can select in newsletters and issues.

```
class ReceiversGroupFilterInactiveOrganizations(object):
    """ Filter all inactive organizations, out of the group selection list.
        receivers: [(id, {'email': 'info@example.com',...})]
    """

    def __init__(self, context):
        self.context = context

    def filter(self, receivers):
        portal = getSite()
        query = {}
        query['portal_type'] = ['Organization']
        query['review_state'] = ['inactive', 'internal', 'pending', 'former_member']
        inactive_groups = portal.membrane_tool.search(query)
        blacklist = [black.getGroupId for black in inactive_groups]
        receivers = [receiver for receiver in receivers
                if receiver[0] not in blacklist]
        return receivers
```

This filter should be registered as follow:

```
<subscriber zcml:condition="installed my.package"
    for="Products.EasyNewsletter.interfaces.IEasyNewsletter"
    factory="my.package.newsletter.ReceiversGroupFilterInactiveOrganizations"
    provides="Products.EasyNewsletter.interfaces.IReceiversGroupFilter" />
```

`Interface: Products.EasyNewsletter.interfaces.IReceiversMemberFilter`

The IReceiversPostSendingFilter can be used to filter the list of reveivers before sending emails to all receivers.

```
class ReceiversPostSendingFilterNoNewsletter(object):
    """ Filter all contacts that has not set the receive_newsletter
    flag, out of receivers email list. But only if the Newsletter provide
    IReceiversMemberFilterNoNewsletter.
    receivers: [{'email': 'info@example.com',...}]
    """

    def __init__(self, context):
        self.context = context

    def filter(self, receivers):
        newsletter_object = self.context
        if IReceiversMemberFilterNoNewsletter.providedBy(newsletter_object):
            portal = getSite()
            query = {}
            query['portal_type'] = ['Contact']
            query['getReceive_newsletter'] = False
            no_enl_contacts = portal.membrane_tool.search(query)
```

(continues on next page)

(continued from previous page)

```
            blacklist = [black.getUserId for black in no_enl_contacts]
            receivers = [receiver for receiver in receivers
                    if receiver['email'] not in blacklist]
        return receivers
```

This filter should be registered as follow:

```
<subscriber zcml:condition="installed my.package"
    for="Products.EasyNewsletter.interfaces.IEasyNewsletter"
    factory="my.package.newsletter.ReceiversPostSendingFilterNoNewsletter"
    provides="Products.EasyNewsletter.interfaces.IReceiversPostSendingFilter" />
```

2.9 Configuring external subscriber sources

An external subscriber sources provides (additional) subscriber to a newsletter instance.

You configure an external subscriber source as a Zope 3 utility providing ISubscriberSource (here's an example where subscriptions are managed externally through MongoDB):

```
class NewsletterSource(object):

    implements(ISubscriberSource)

    def getSubscribers(self, newsletter):
        """ return all subscribers for the given newsletter
            from the MyInfo user database. Newsletter subscriptions
            are referenced inside MyInfo through UIDs.
        """

        uid = newsletter.UID()

        # find MyInfo subscribers
        myinfo = getUtility(IMyInfo)
        subscribers = list()
        for user in myinfo.accounts.find({'data.newsletters': uid, 'state': 'active'}
↪):
            subscribers.append(dict(email=user['email'], fullname=user['username']))
        return subscribers
```

The utility must be registered using ZCML:

```
<utility zcml:condition="installed Products.EasyNewsletter"
    name="MyInfo subscribers"
    factory=".newsletter.NewsletterSource"
    />
```

Inside the `Edit` view of the instance under the `External` tab you should find `MyInfo subscribers` under the option `External subscriber source`.

CHAPTER 3

Allowed placeholders

The following placeholder can be used in the header, body and footer fields or the aggregation and output templates:

- `{{SUBSCRIBER_SALUTATION}}` example: Dear Ms.
- `{{salutation}}` example: Ms.
- `{{unsubscribe}}` unsubscribe link to be included in emails
- `{{receiver}}` example: `{'salutation': 'Guten Tag', 'nl_language': 'de', 'fullname': 'Test Member', 'email': 'maik@planetcrazy.de'}`
- `{{language}}` example: de
- `{{fullname}}`
- `{{issue_title}}`
- `{{issue_description}}`
- `{{banner_src}}` Banner src url
- `{{logo_src}}` Logo src url
- `{{date}}` example: 30.05.2017
- `{{month}}` example: 5
- `{{year}}` example: 2017

CHAPTER 4

Customize personalization (placeholders)

You can use the BeforePersonalizeEvent to override placeholder values or even manipulate the html before the placeholder are filled.

First define a event handler somewhere in you package.

```
def enl_personalize(event):
    edc = event.data['context']
    event.data['html'] = event.data['html'].replace('PHP', 'Python')
    firstname = edc['receiver'].get('firstname')
    lastname = edc['receiver'].get('lastname')
    if not firstname and not lastname:
        edc['SUBSCRIBER_SALUTATION'] = u'Dear {0}'.format(
            edc['receiver']['email']
        )
```

Then register the event handler for the BeforePersonalizeEvent:

```
<subscriber
  for="Products.EasyNewsletter.interfaces.IBeforePersonalizationEvent"
  handler=".subscribers.enl_personalize"
  />
```

For a working example see `test_before_the_personalization_filter` in `test_newsletter.py`.

CHAPTER 5

Aggregation templates

We provide some content aggregation templates, but you can add more. A aggregation template has to be global available template. The default templates are in the skins folder. They have to be registered in the Plone registry, see below for an example how to do that with GenericSetup.

You can add your aggregation templates TTW in /portal_skins/custom/manage_main and add them to the registry in /portal_registry. Search there for EasyNewsletter, you will find `Products EasyNewsletter content_aggregation_templates` where you can add your templates.

To do it with your addon product, add this to your registry.xml in your profiles.

```
<record name="Products.EasyNewsletter.content_aggregation_templates">
    <field type="plone.registry.field.Dict">
        <title>ENL Content aggregation templates</title>
        <key_type type="plone.registry.field.TextLine" />
        <value_type type="plone.registry.field.TextLine" />
    </field>
    <value purge="false">
        <element key="aggregation_fancy_pictures">Fancy pictures listing</element>
    </value>
</record>
```

CHAPTER 6

Output templates

We provide some output templates, but you can add more. A output template has to be global available template. The default templates are in the skins folder. They have to be registered in the Plone registry, see below for an example how to do that with GenericSetup.

You can add your output templates TTW in /portal_skins/custom/manage_main and add them to the registry in /portal_registry. Search there for EasyNewsletter, you will find `Products EasyNewsletter output_templates` where you can add your templates.

To do it with your addon product, add this to your registry.xml in your profiles.

```
<record name="Products.EasyNewsletter.output_templates">
    <field type="plone.registry.field.Dict">
        <title>ENL Output templates</title>
        <key_type type="plone.registry.field.TextLine" />
        <value_type type="plone.registry.field.TextLine" />
    </field>
    <value purge="false">
        <element key="output_green_energy">Green energy output template</element>
    </value>
</record>
```

CHAPTER 7

Documentation

Online documentation: http://productseasynewsletter.readthedocs.io

CHAPTER 8

Source Code

- repository: https://github.com/collective/Products.EasyNewsletter

CHAPTER 9

Bugtracker

- https://github.com/collective/Products.EasyNewsletter/issues

CHAPTER 10

Authors

- initial release: Kai Dieffenbach
- Maik Derstappen
- Andreas Jung
- Philip Bauer
- Timo Stollenwerk
- Dinu Gherman
- Jens Klein
- Peter Holzer

Contents:

CHAPTER 11

Indices and tables

- genindex
- modindex
- search