gs.profile.notify Documentation

Release 3.4.0

GroupServer.org

July 08, 2016

Contents

Author Michael JasonSmith

Contact Michael JasonSmith <mpj17@onlinegroups.net>

Date 2015-06-24

Organization GroupServer.org

Notifications are email messages that GroupServer are sends outside the normal group-email context. In this document I list all the notifications in GroupServer, discuss how to write a notification, and document the `MessageSender` class, which sends a message to a person.

Contents:

CHAPTER 1

GroupServer Notifications

Authors Alice Murphy; Michael JasonSmith

Contact Michael JasonSmith <mpj17@onlinegroups.net>

Organization GroupServer.org

Table of contents

Notifications are small messages that are sent to the user outside the group-email context. They include messages such as the *invitation* to join a group, and the *group welcome* email. In this document we describe the various notifications sent by GroupServer. We summarise who they are sent to, and the code-path that is followed to send the notification.

Currently the notifications are being transitioned to their third version. These *file-system-side notifications* are are more flexible than the old *notification templates*. The notifications in different sub-systems will be converted to file-system side notifications when each module is reviewed in the normal process of software maintenance and refactoring.

1.1 File-System-Side Notifications

As we rebuild each subsystem, we move its notifications to the file system, from the ZMI. The file-system-side notifications work like Web pages. The notification system renders two pages — an HTML page and a plain-text version of the same message — and places them in an email message. All of the new fine-system-side notifications use `gs.profile.notify.sender.MessageSender` to send the message.

1.1.1 Cannot Post

Sent to a person with a profile when they attempt to post to a group, but disallowed. The content of the message comes from the viewlets that make up the `gs.group.type.*` eggs. For people people without a profile see *unknown email address*.

Sent to The a person with a profile that cannot post to the group.

URL *Group Page* `/cannot-post.html`

via

`Products.XWFMailingListManager.XWFMailingList.checkMail`
`gs.group.member.canpost.notifier.Notifier`
`gs.group.member.canpost.notifier.CannotPostMessageSender`
`gs.profile.notify.notifyuser.NotifyUser`

1.1.2 Invitation

An invitation is a message from an administrator asking someone to join the group. It is also used to present the administrator with a *preview* of the invitation.

Sent to Someone who has been invited to join a group by the administrator.

URL *Group page* `/invitationmessage.html`

via Clicking *Invite*

`gs.group.member.invite.base.invite.InviteEditProfileForm`
`gs.group.member.invite.base.processor.InviteProcessor`
`gs.group.member.invite.base.inviter.Inviter`
`gs.group.member.invite.base.notify.InvitationNotifier`
`gs.profile.notify.sender.MessageSender`

Invite site member:
`gs.group.member.invite.invitesitemembers.GSInviteSiteMembersForm`
`gs.group.member.invite.json.api.InviteUserAPI`
`gs.group.member.invite.base.processor.InviteProcessor`
...

Invite in bulk: `gs.group.member.invite.csv.ui.CSVUploadUI`
`gs.group.member.invite.json.api.InviteUserAPI`
`gs.group.member.invite.base.processor.InviteProcessor`
...

Resend: `gs.group.member.invite.resend.reinvite.ResendInvitationForm`
`gs.group.member.invite.base.processor.InviteProcessor`
...

1.1.3 Group Welcome

The *Group Welcome* notification is sent to a new member when he or she joins a group. However, there are many ways of becoming a member, and some still use the old code, rather than this shiny method.

Site administrators receive the *New Member* notification.

Sent to A new member of a group.

URL *Group page* `/new-member-msg.html`

via A *logged in member* clicks `Join` in a Public group.

`gs.group.join.join.JoinForm`
`gs.group.join.notify.NotifyNewMember`
`gs.profile.notify.sender.MessageSender`

A new **invited** member accepts an invitation to join a group
`gs.profile.invite.initalresponse`

An existing **invited** member accepts an invitation to join a group.
`gs.profile.invite.invitationsrespond`

A new member joins a group during **registration**
`gs.profile.signup.base.changeprofile.ChangeProfileForm`
or `gs.profile.signup.base.verifywait.VerifyWaitForm`

1.1.4 Group Started

Information about the group that has just been started

Sent to Every site administrator.

URL *Group page* `/gs-group-start.html`

via A *site administrator* clicks `Start`

`gs.group.start.startgroup.StartGroupForm`
`gs.group.start.notify.StartNotifier`
`gs.profile.notify.sender.MessageSender`

1.1.5 New Member

Sent to the group administrators when a new member joins the group. It is the flip-side of the *group welcome* notification.

Sent to The group administrators, or the site administrators if there are no group administrators.

URL *Group page* `/new-member-admin-msg.html`

via A *logged in member* clicks `Join` in a Public group.

`gs.group.join.join.JoinForm`
`gs.group.join.notify.NotifyAdmin`
`gs.profile.notify.sender.MessageSender`

A new **invited** member accepts an invitation to join a group
`gs.profile.invite.initalresponse`

An existing **invited** member accepts an invitation to join a group
`gs.profile.invite.invitationsrespond`

An administrator accepts the *request* to join the group. #3469
`gs.group.member.request.request.respond.Respond`

A new member joins a group during **registration**
`gs.profile.signup.base.changeprofile.ChangeProfileForm`
or `gs.profile.signup.base.verifywait.VerifyWaitForm`

1.1.6 Request Membership

This message is sent when someone requests to become a member of a Private group. It is the opposite of a *invitation*. It should not be confused with *Request Contact*.

Sent to The administrator of the group.

URL *Group page* `/request_message.html`

via The request membership form | `gs.group.member.request.request.RequestForm` | `gs.profile.notify.sender.MessageSender`

1.1.7 Request Contact

This notification is sent when a member reaches out to another. It allows the email address of everyone to be kept secret until they chose to disclose it. It is unusual because the *`From`* and *`Reply-to`* addresses are different.

Sent to The person being contacted.

URL *Profile page* `/request_contact.html`

via The request contact form | `gs.profile.contact.request.RequestContact` | `gs.profile.contact.notify.RequestNotifier` | `gs.profile.contact.notify.AlternateReplyMessageSender` | `gs.profile.notify.sender.MessageSender`

1.1.8 Reset password

A link to reset a password, sent to an email address that is submitted via the *Reset Password* page, when the email address is recognised as belonging to a user.

Sent to The person that requested the password reset.

URL `gs-profile-password-reset-message.html` in the context of a user.

via

`gs,profile.password.request.RequestPasswordResetForm`
`gs,profile.password.notifier.ResetNotifier`
`gs.profile.notify.sender.MessageSender`

1.1.9 Topic digest

The topic digest contains a summary of the topics that were discussed recently in the group. A "cron-job" is used to regularly send out the digests, using the `senddigest` command. The digest system consists of two notifications: *the daily digest*, and *the weekly digest*. In addition there are two commands: the *digest on command*, and the *digest off command*.

The daily digest

The daily digest of topics topic digest is sent every day when there are posts. The digest

Sent to All group members who have elected to receive posts in digest form.

URL *Group Page* `gs-group-messages-topic-digest-daily.html`

via

`gs.group.messages.topic.digest.send.script.main`
`gs.group.messages.topic.digest.send.script.send_digest`
Site page `gs-group-messages-topic-digest-send.html`
`gs.group.messages.topic.digest.base.sendDigests.SendDigests`
[`gs.group.messages.topic.digest.daily.notifier.DailyDigestNotifier`]
`gs.group.messages.topic.digest.base.notifier.DigestNotifier.notify`
`gs.email.send_email()`

The weekly digest

The weekly digest is sent once a week, on the weekly-anniversary of the last post, if there have been no posts that week.

Sent to All group members who have elected to receive posts in digest form.

URL *Group Page* `gs-group-messages-topic-digest-weekly.html`

via

`gs.group.messages.senddigest.script.main`
`gs.group.messages.senddigest.script.send_digest`
Site page `gs-group-messages-topic-digest-send.html`
`gs.group.messages.topic.digest.base.sendDigests.SendDigests`

[`gs.group.messages.topic.digest.weekly.notifier.WeeklyDigestNotifier`]
`gs.group.messages.topic.digest.base.notifier.DigestNotifier.notify`
`gs.email.send_email`

Digest on command

There is an email-command to turn the digest on. It is triggered when a group member sends an email to the group with the subject `digest on` (case insensitive).

Sent to The person that asked for the digest to be turned on

URL `gs-group-member-email-settings-digest-on.html` in the context of a group.

via

`gs.group.member.email.settings.listcommand.DigestCommand`
`gs.group.member.email.settings.notifier.DigestOnNotifier`
`gs.profile.notify.sender.MessageSender`

Digest off command

There is an email-command to turn the digest on. It is triggered when a group member sends an email to the group with the subject `digest on` (case insensitive).

Sent to The person that asked for the digest to be turned on

URL `gs-group-member-email-settings-digest-off.html` in the context of a group.

via

```
gs.group.member.email.settings.listcommand.DigestCommand
gs.group.member.email.settings.notifier.DigestOffNotifier
gs.profile.notify.sender.MessageSender
```

1.1.10 Unknown Email Address

A post is received by the mailing list from an unregistered email address. It is the equivalent of the *cannot post* notification for anonymous people.

Sent to The unrecognised email address, which sent the original message.

URL *Group Page* `/unknown-email.html`

via

```
Products.XWFMailingListManager.XWFMailingList.processMail
Products.XWFMailingListManager.XWFMailingList.mail_reply
gs.group.member.canpost.unknownemail.Notifier

Products.XWFMailingListManager.XWFMailingList.requestMail
Products.XWFMailingListManager.XWFMailingList.mail_reply
gs.group.member.canpost.unknownemail.Notifier

Products.XWFMailingListManager.XWFMailingList.processModeration
Products.XWFMailingListManager.XWFMailingList.mail_reply
gs.group.member.canpost.unknownemail.Notifier
```

1.1.11 Verify Email Address

Email addresses must be verified. The verification message is sent from everywhere that email addresses can be added. It turns out that there are *many* places that an email address can be added. The method `gs.profile.email.verify.emailverificationuser.EmailVerificationUser.send_verification` sends the verification message for all higher-level code.

Sent to The person who has the new address.

URL *Profile page* `/verification-mesg.html`

via Anywhere that lets the user add an email address

Registering as a new user (or requesting membership as a new user)

```
gs.profile.signup.base.request_registration.RequestRegistrationForm
```

```
gs.profile.email.verify.emailverificationuser.EmailVerificationUser
gs.profile.email.verify.notify.Notifier
gs.profile.notify.sender.MessageSender
```

Adding a new email address, or sending another verification message during registration
`gs.profile.signup.base.verifywait.VerifyWaitForm`

Adding a new email address
`gs.profile.email.settings.settings.ChangeEmailSettingsForm`

1.1.12 Bounce

When GroupServer gets an XVERP return it logs a bounce. If the group member has another email address then the user is told of the bounce on the extra address.

Sent to The person who has the bouncing address

URL *Group page* `/gs-group-member-bounce-bouncing.html`

via The *Handle bounce* page

```
gs.group.member.bounce.handlebounce.HandleBounce
gs.group.member.bounce.notifier.UserBounceNotifier
gs.profile.notify.sender.MessageSender
```

1.1.13 Disabled

When an address continually bounces then the address is disabled.

Disabled (user)

The user is told of that an address is disabled if he or she has an extra address.

Sent to The person who has the bouncing address

URL *Group page* `/gs-group-member-bounce-disabled.html`

via The *Handle bounce* page

```
gs.group.member.bounce.handlebounce.HandleBounce
gs.group.member.bounce.notifier.UserDisabledNotifier
gs.profile.notify.sender.MessageSender
```

Disabled (administrator)

The administrator is told when a member has his or her email address disabled because of bouncing.

Sent to The administrators of the group that sent the post that bounced back.

URL *Group page* `/gs-group-member-bounce-disabled-admin.html`

via The *Handle bounce* page

```
gs.group.member.bounce.handlebounce.HandleBounce
gs.group.member.bounce.notifier.AdminDisabledNotifier
gs.profile.notify.sender.MessageSender
```

1.1.14 Leave

Like joining, the member and the administrators are told that someone has left a group. A person can leave in two ways: using the Web or sending an email with the subject `unsubscribe` (case insensitive) to the group.

Leave (past member)

Sent to The person who has just left a group

URL *Group page* `/gs-group-member-leave-notification.html`

via The *Leave* page

```
gs.group.member.leave.base.leave.LeaveForm
gs.group.member.leave.base.leave_group()
gs.group.member.leave.base.notifier.LeaveNotifier
gs.profile.notify.sender.MessageSender
```

via The *Unsubscribe* command

```
gs.group.member.leave.command.LeaveCommand
gs.group.member.leave.base.leave_group()
gs.group.member.leave.base.notifier.LeaveNotifier
gs.profile.notify.sender.MessageSender
```

Leave (administrator)

Sent to The administrators of a group from which a person has just left.

URL *Group page* `/gs-group-member-leave-left.html`

via The *Leave* page

```
gs.group.member.leave.base.leave.LeaveForm
gs.group.member.leave.base.leave_group()
gs.group.member.leave.base.notifier.LeftNotifier
gs.profile.notify.sender.MessageSender
```

via The *Unsubscribe* command

```
gs.group.member.leave.command.LeaveCommand
gs.group.member.leave.base.notifier.LeftNotifier
gs.profile.notify.sender.MessageSender
```

Not a member

If someone tries to leave, but the email address in the `From` header does not match then a special *Not a Member* email is sent.

Sent to The person who has asked to leave a group

URL *Groups* `/gs-group-member-leave-not-a-member.html` (**Note** not the *group* page.)

via The *Unsubscribe* command

```
gs.group.member.leave.command.LeaveCommand
gs.group.member.leave.command.notifiernonmember.NotMemberNotifier
gs.profile.notify.sender.MessageSender
```

1.1.15 Profile status

The monthly profile-status notification is sent out monthly to everyone.

Sent to Every person that is in at least one group in the GroupServer install.

URL *Site page* `/gs-profile-status.html`

via

```
gs.profile.status.send.script.main
gs.group.messages.topic.digest.send.script.send_status
    Site page /gs-profile-status.html
gs.profile.status.base.hook.SendNotification
gs.profile.status.base.notifier.StatusNotifier
gs.profile.notify.sender.MessageSender
```

1.2 Notification Templates

These are the old notifications. They are DTML templates: this is the folder in which `Products.CustomUserFolder.Customuser.send_notification` looks to find the notifications passed to it by ID.

1.2.1 Moderation

Moderation is a world unto its own, and is badly need of a rewrite [1].

[1] *Ticket 249: Rebuild Moderation* summarises the problems with moderation, and how to fix it <https://projects.iopen.net/groupserver/ticket/249>

mail_moderated_user

A message to the group is received from a moderated member.

Sent to

The moderated member.

via

```
Products.XWFMailingListManager.XWFMailingList.processMail
Products.XWFMailingListManager.XWFMailingList.processModeration
Products.CustomUserFolder.CustomUser.send_notification
```

mail_moderator

A message to the group is received from a moderated member.

Sent to

The moderators.

via

```
Products.XWFMailingListManager.XWFMailingList.processMail
Products.XWFMailingListManager.XWFMailingList.processModeration
Products.CustomUserFolder.CustomUser.send_notification
```

CHAPTER 2

How to Write a Notification

Or, more correctly, how Michael writes notifications.

I write notifications in three steps: *1. Write an HTML Page*, *2. Write a Text Page*, and *3. Write a Notifier*.

2.1 1. Write an HTML Page

A *MessageSender* takes both a HTML and a plain-text version of a message. However, I find it easier to start with the HTML form, and then work on the text-version (see *2. Write a Text Page* below). I write the HTML form of the message as a normal page-view.

If the message is **about a group,** or a group member, I will make the view a subclass of the `gs.content.email.base.GroupEmail` class [1], so it is in the group context [2].

```
class InvitationMessage(GroupEmail):

    def __init__(self, context, request):
        super(InvitationMessage, self).__init__(context, request)
```

Messages about group members are placed in the group-context so the permissions are correct.

```
<browser:page
  name="invitationmessage.html"
  for="gs.group.base.interfaces.IGSGroupMarker"
  class=".notifymessages.InvitationMessage"
  template="browser/templates/new-invitationmessage.pt"
  permission="zope2.View" />
```

If the message is *just* **about an individual,** removed from the context of the group, I will make the page a subclass of the `gs.content.email.base.SiteEmail` class [1].

```
from gs.content.email.base import SiteEmail, TextMixin

class ProfileStatus(SiteEmail):
    'The profile-status notification'
```

The page is rendered in the context of a user it is in the context of a user [4].

[1] See `gs.content.email.base` <https://github.com/groupserver/gs.content.email.base>
[2] A page in the group-context will hang off the `gs.group.base.interfaces.IGSGroupMarker` marker interface
[4] A page in the context of a user will hang off the marker interface `Products.CustomUserFolder.interfaces.ICustomUser`.

```
    <browser:page
      name="gs-profile-status.html"
      for="Products.CustomUserFolder.interfaces.ICustomUser"
      class=".notification.ProfileStatus"
      template="browser/templates/notification-html.pt"
      permission="zope2.ManageProperties" />
```

The view sometimes has code specific to the message, just like other page-views often have page-specific code in them.

At the top of the page-template I set up all the arguments to the page. Later these will be passed in as *options* (see *3. Write a Notifier* below). However, for prototyping it is easier to use hard-coded defaults if the arguments are not supplied:

```
<html
  tal:define="userInfo options/userInfo | view/userInfo;
              emailAddress options/emailAddress | string:placeholder@email.address;
              verifyLink options/verifyLink | string:${view/siteInfo/url}/r/verify/placeholder">
```

The above code is in the context of a user, so there is always a `view/userInfo` available for the `userInfo` option. In the group-context I use the logged-in user information (`self.loggedInUserInfo`) to fill in the user-specific details. The two other options use place-holder strings: one completely hard-coded, and one with some site-specific information.

2.2 2. Write a Text Page

The text-page is normally a cut-down version of the HTML-page. It hangs off the same marker interfaces, and I will give the page the same name, except with a `.txt` extension.

I normally make the view a subclass of the HTML view, and make use of the `from gs.content.email.base.TextMixin` class.

```
class SomeNotificationText(SomeNotificationHTML, TextMixin):

    def __init__(self, context, request):
        super(SomeNotificationText, self).__init__(context, request)
        filename = 'some-notification-{0}.txt'.format(self.groupInfo.id)
        self.set_header(filename)

    def format_message_no_indent(self, m):
        tw = TextWrapper()
        retval = tw.fill(m)
        return retval
```

The page-template itself normally follows the HTML closely, but with all styling removed and the remaining elements replaced with `<tal:block />` elements.

2.3 3. Write a Notifier

A notifier is what is called by the UI to send the message. It creates the HTML and text forms of the message, and sends it to the correct people. Normally they notifier is a sub-class of the `gs.content.email.base.NotifierABC` abstract base-class, or one of its subclasses [1]:

```
class DigestOnNotifier(GroupNotifierABC):
    htmlTemplateName = 'gs-group-member-email-settings-digest-on.html'
    textTemplateName = 'gs-group-member-email-settings-digest-on.txt'

    def notify(self, userInfo):
        subject = _('digest-on-subject',
                    'Topic digests from ${groupName}',
                    mapping={'groupName': self.groupInfo.name})
        translatedSubject = translate(subject)
        text = self.textTemplate()
        html = self.htmlTemplate()

        sender = MessageSender(self.context, userInfo)
        sender.send_message(translatedSubject, text, html)
        self.reset_content_type()
```

Initialisation: The notifier needs access to both the `request` and `context`. Because of this it is the responsibility of the user-interfaces (normally forms) to send the notifications. It is not the responsibility of the low-level code that actually does the work.

`htmlTemplate`: This *property* acquires the HTML view of the message. To do this it calls:

```
    getMultiAdapter((self.context, self.request),
                    name=self.htmlTemplateName)
```

This is the same way that the normal publishing system acquires the view for display. The view is not rendered until the `notify` method is called (see below).

`textTemplate`: This *property* works much the same way as the HTML Template property, but the name of the text-view is passed in.

`notify`: This **method** does three things.

1. Renders the HTML and text versions of the message. It does this by passing in any options that are needed by the page. For example:

```
        text = self.textTemplate(userInfo=userInfo)
        html = self.htmlTemplate(userInfo=userInfo)
```

2. Instantiating the *`MessageSender`* class.
3. Calling the `MessageSender.send_message()` method.

The main difference between the different Notifier classes are different views are created (the names passed to the named-adaptor calls are different), and the `notify` method takes different arguments. These arguments are normally blindly passed on to the two views.

CHAPTER 3

gs.profile.notify API

The *MessageSender* class is the main symbol provided by the gs.profile.notify product.

class gs.profile.notify.**MessageSender**(*context*, *userInfo*)

Parameters

- **context** – The context of the message.
- **userInfo** (Products.CustomUserFolder.interfaces.IGSUserInfo) – The person to send the message to.

The *MessageSender* is used to send a pre-written message to someone. To *initialise* the class pass in a context and a user-info for the person who is to receive the message.

send_message(self, subject, txtMessage, htmlMessage='', fromAddress=None, toAddresses=None)

Parameters

- **subject** (*str*) – The subject (the *Subject* header) of the message.
- **txtMessage** (*str*) – The plain-text (*text/plain*) version of the message.
- **htmlMessage** (*str*) – The HTML version (*text/html*) of the message.
- **fromAddress** (*str*) – The address the email is sent *from* (the *From* header).
- **toAddress** (*list or None*) – The addresses the email is sent *to* (the *To* header).

•If only the subject and txtMessage arguments are given then the message will be sent to the default email addresses of the user that was passed in when the message sender was initialised.

•If the optional htmlMessage is provided then a *multipart/alternative* email message will be created (**RFC 2046#section-5.1.4**), with both the text and HTML forms of the message set.

•The fromAddress sets who sent the message. If omitted the email address of the *Support* group is used [1].

•Finally, the toAddress is a list of email addresses to send the notification to. If omitted the system will use default (preferred) addresses of the user that was passed in when the message sender was initialised [2].

[1] The system will fail an assertion if it cannot find a user for the supplied fromAddress.

[2] The system will fail an assertion if a toAddress is used that does not belong to the user. The address may be *unverified*, but it must belong to the user.

The *MessageSender* does not, ultimately, send the message. Instead it formats the message [3], and then calls `gs.profile.notify.notifyuser.NotifyUser`. The `gs.profile.notify.notifyuser.NotifyUser.send_message()` method of this class sends the message on its way by calling the `gs.email.send_email()` function.

[3] The core Python `email` module is used to format the message using MIME.

CHAPTER 4

Changelog

4.1 3.4.0 (2016-07-08)

- Making `MessageSender.set_headers` its own method
- Updating the list of Notifications, because *Request contact* is now a file-system side notification

4.2 3.3.1 (2016-02-10)

- Handling missing `From` addresses better
- Adding some unit tests

4.3 3.3.0 (2015-06-24)

- Moving the documentation to Sphinx and Read the Docs

4.4 3.2.2 (2015-02-17)

- Added some Unicode robustness

4.5 3.2.1 (2015-02-04)

- Added a `Date` header

4.6 3.2.0 (2014-09-23)

- Switching to GitHub as the primary repository
- Using the `@implementer` class decorator rather than the `implements` statement
- Unicode fixes to the audit trail
- Updating the documentation

4.7 3.1.3 (2014-06-11)

- Added more error checking

4.8 3.1.2 (2014-02-28)

- Updated the `To` and `From` headers
- Updated the documentation on the *Reset password* notification

4.9 3.1.1 (2014-01-29)

- Updated documentation

4.10 3.1.0 (2013-10-21)

- Better auditing
- Ensuring that the `gs.profile.notifiy.NotifiyUser` is exposed to ZMI-side stripts

4.11 2.1.0 (2012-06-22)

- Update to SQLAlchemy

4.12 2.0.3 (2012-03-27)

- Updating the documentation

4.13 2.0.2 (2012-02-07)

- Using the `@Lazy` decorator in the message sender

4.14 2.0.1 (2012-01-18)

- Added to the documentation
- Fixing some context issues

4.15 2.0.0 (2011-05-18)

- Added the ability to send HTML formatted email notifications
- Made the `From` address optional

4.16 1.1.0 (2011-01-18)

- Added an `EmailUser` class
- Following the gs.profile.email.base code

4.17 1.0.2 (2010-09-28)

- Getting the site-information more reliably
- Fixing an overly zealous assert

4.18 1.0.1 (2010-07-09)

- Removing a `<five:implements>` declaration from the ZCML

4.19 1.0.0 (2010-04-15)

Initial version. Prior to the creation of this product email notifications were handled by Products.CustomUserFolder and templates in the `Templates/email/notifications` folder of the ZMI. The `NotifyUser` code was originally written by was written by Richard Waid.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

CHAPTER 6

Resources

- Documentation: http://groupserver.readthedocs.io/projects/gsprofilenotify/
- Code repository: https://github.com/groupserver/gs.profile.notify
- Questions and comments to http://groupserver.org/groups/development
- Report bugs at https://redmine.iopen.net/projects/groupserver

M

R