# gs.group.list.command Documentation

***Release 1.0.3***

**GroupServer.org**

December 11, 2015

Contents

Contents:

CHAPTER 1

# gs.group.list.command API Reference

The API for email-commands is in two parts: *processing commands*, and *the result enumeration*.

## 1.1 Processing commands

The *gs.group.list.command.process_command()* function is used to process the commands in an email message.

gs.group.list.command.**process_command**(*group*, *email*, *request*)

Process a command in an email message

**Parameters**

- **group** (*obj*) – The group that recieved the email message.
- **email** (str or email.message.Message) – The email message that was recieved (which may or may not contain a command).
- **request** (*obj*) – The current browser request object.

**Returns** If a command was processed, and if email processing should continue.

**Return type** *CommandResult*

When an email is recieved it needs to checked to see if its Subject header is command, and the command executed if necessary. The *process_command()* function performs both of these tasks. The result will be either

- CommandResut.notACommand if the email is a normal message,
- CommandResut.commandStop if the email contained a command and processing should stop, or
- CommandResut.commandContinue if the email contained a command and processing should continue.

### 1.1.1 Example

```
r = process_command(self.group, email, request)
if r == gs.group.list.command.CommandResult.commandStop:
  return
```

## 1.2 Command Abstract Base Class

The `CommandABC` abstract base-class provides some useful functionality

**class** `gs.group.list.command.`**CommandABC**(*group*)
Abstract base-class for command-adaptors

**Parameters** **group** (*object*) – The group that is adapted.

**static** **get_command_components**(*email*)
Get the components of the command in the `Subject`

**Parameters** **email** (`email.message.Message`) – The email message that contains the command.

**Returns** The `Subject` of the email, split into components and lower-cased.

**Return type** list of strings

The `get_command_components()` method splits the command in the `Subject` into parts using the `shlex.split()` function. The components of the command are in lower-case, with all `re:` parts discarded.

**process**(*email*, *request*)
Process the command in the email

**Parameters**

- **email** (`email.message.Message`) – The email message that contains the command.
- **request** – The HTTP request made to process the email.

**Returns** If a command was processed, and if email processing should continue.

**Return type** *CommandResult*

*Concrete* classes must implement this method.

Sub-classes of `CommandABC` will need to provide the `process()` method. The browser-request is passed in so the command can issue email-notifications.

## 1.3 The Result Enumeration

The result enumeration is returned by the *`gs.group.list.command.process_command()`* function, and the command that are registered.

**class** `gs.group.list.command.`**CommandResult**
An enumeration of the different results from processing a command.

**commandContinue = <CommandResult.commandContinue: 2>**
The command was processed, and processing of this email should continue.

**commandStop = <CommandResult.commandStop: 1>**
The command was processed, and processing of this email should stop.

**notACommand = <CommandResult.notACommand: 0>**
The `Subject` did not contain a command

CHAPTER 2

# Changelog

## 2.1 1.0.3 (2015-12-11)

- Fixing the unit tests (possibly an issue with Zope2 2.13.23)

## 2.2 1.0.2 (2015-01-26)

- Tweaking the MANIFEST

## 2.3 1.0.1 (2014-11-13)

- Dealing with subject lines that only have one quote
- Dealing with subject lines that are missing, empty, or blank

## 2.4 1.0.0 (2014-10-01)

- Initial release

Prior to the creation of this product the command processing was carried out in the `Products.XWFMailingListManager.XWFMailingList` class.

CHAPTER 3

# Introduction

This product provides support for email-commands. It does this by providing a function for processing commands (to check for a command in an email message), a way to *register command processors*, and the result enumeration for returning the result of a command.

CHAPTER 4

# Register command processors

The commands are named *adaptors* that implement the `gs.group.list.command.interfaces.IEmailCommand` interface. The *name* is the command-name in **lower case.** So the command to unsubscribe someone from a group will have the adaptor name `unsubscribe`. The adaptor must

- Take the group in the `__init__()` method (it adapts the group),
- Provide a `process()` method that takes the email and browser-request as an argument.

## 4.1 Example

I prefer to declare adaptors using ZCML. This will declare a command named `example`. This command will be executed by `process_command()` whenever the subject line of an email message contains starts with `example` (in upper or lower case). The command itself is implemented by the `ExampleCommand` class in the `example` module in the local directory:

```
<adapter
  name="example"
  for="gs.group.base.interfaces.IGSGroupMarker"
  provides="gs.group.list.command.interfaces.IEmailCommand"
  factory=".example.ExampleCommand" />
```

The `example` module would contain the `ExampleCommand` class, which inherits from the abstract base-class for commands.

```
from gs.group.list.command import CommandABC, CommandResult
class ExampleCommand(CommandABC):
    def process(email, request):
        # TODO: Stuff
        return CommandResult.commandStop
```

The `request` is passed in to the `process()` method so the class can issue email-notifications.

CHAPTER 5

# Indices and tables

- genindex
- modindex
- search

## C



## G



## N



## P