asphalt-filewatcher

Release 0.0.post11

Sep 27, 2017

Contents

This Asphalt framework component provides means for applications to watch files and directories and receive notifications for changes made to them.

A wide variety of mechanisms are supported:

- inotify (Linux)
- ReadDirectoryChangesW (Windows)
- FSEvents (Mac OS X)
- kqueue (*BSD, Mac OS X)
- periodic polling (all platforms)

CHAPTER 1

Configuration

To configure a mailer for your application, you need to choose a backend and then specify any necessary configuration values for it. The following backends are provided out of the box:

- `smtp` (**recommended**)
- `sendmail`
- `mock` (for testing only)

Other backends may be provided by other components.

Once you've selected a backend, see its specific documentation to find out what configuration values you need to provide, if any. Configuration values are expressed as constructor arguments for the backend class:

```
components:
  mailer:
    backend: smtp
    host: primary-smtp.company.com
    ssl: true
    username: foo
    password: bar
```

This configuration uses `primary-smtp.company.com` as the server hostname and uses implicit TLS to encrypt the connection. It authenticates with the server using the username `foo` and the password `bar`.

The above configuration can be done directly in Python code as follows:

```
class ApplicationComponent(ContainerComponent):
    async def start(ctx: Context):
        self.add_component(
            'mailer', backend='smtp', host='primary-smtp.company.com', ssl=True,
            username='foo', password='bar')
        await super().start()
```

Multiple mailers

If you need multiple mailers, you need to specify them via the `mailers` argument, which is a dictionary of resource names to their backend configuration options:

```
components:
  mailer:
    mailers:
      smtp_a:
        backend: smtp
        context_attr: mailer1
        host: primary-smtp.company.com
        ssl: true
        username: foo
        password: bar
      smtp_b:
        backend: smtp
        context_attr: mailer2
        host: isp-smtp.provider.com
      sendmail:
        backend: sendmail
        context_attr: mailer3
```

This configures three mailer resources, named `smtp_a`, `smtp_b` and `sendmail`. Their corresponding context attributes are `mailer1`, `mailer2` and `mailer3`. If you omit the `context_attr` option for a mailer, its resource name will be used.

CHAPTER 2

Using mailers

The primary tools for sending email with asphalt-mailer are the `EmailMessage` class and the `deliver()` method. The workflow is to first construct one or more messages and then using the mailer to deliver them.

Two convenience methods are provided to this end: `create_message()` and `create_and_deliver()`. Both methods take the same arguments, but the former only creates a message (for further customization), while the latter creates and delivers a message in one shot, as the name implies.

Email messages can have plain and/or HTML content, along with attachments. The full power of the new standard library email API is at your disposal.

In addition to the examples below, some runnable examples are also provided in the `examples` directory of the source distribution. The same code is also available on Github.

Simple example

This sends a plaintext message with the body "Greetings from Example!" to `recipient@company.com`, addressed as coming from `Example Person <example@company.com>`:

```
async def handler(ctx):
    await ctx.mailer.create_and_deliver(
        subject='Hi there!', sender='Example Person <example@company.com>',
        to='recipient@company.com', plain_body='Greetings from Example!')
```

HTML content

Users may want to send styled emails using HTML. This can be done by passing the HTML content using the `html_body` argument:

```
async def handler(ctx):
    html = "<h1>Greetings</h1>Greetings from <strong>Example Person!</strong>"
    plain = "Greetings!\n\nGreetings from Example Person!"
```

```
    await ctx.mailer.create_and_deliver(
        subject='Hi there!', sender='Example Person <example@company.com>',
        to='recipient@company.com', plain_body=plain, html_body=html)
```

Note: It is highly recommended to provide a plaintext fallback message (as in the above example) for cases where the recipient cannot display HTML messages for some reason.

Attachments

To add attachments, you can use the handy `add_file_attachment()` and `add_attachment()` methods.

The following example adds the file `/path/to/file.zip` as an attachment to the message. The file will be displayed as `file.zip` with the autodetected MIME type `application/zip`:

```
async def handler(ctx):
    message = ctx.mailer.create_message(
        subject='Hi there!', sender='Example Person <example@company.com>',
        to='recipient@company.com', plain_body='See the attached file.')
    await ctx.mailer.add_file_attachment(message, '/path/to/file.zip')
    await ctx.mailer.deliver(message)
```

If you need more fine grained control, you can directly pass the attachment contents as bytes to `add_attachment()`, but then you will have to explicitly specify the file name and MIME type:

```
async def handler(ctx):
    message = ctx.mailer.create_message(
        subject='Hi there!', sender='Example Person <example@company.com>',
        to='recipient@company.com', plain_body='See the attached file.')
    ctx.mailer.add_attachment(message, b'file contents', 'attachment.txt')
    await ctx.mailer.deliver(message)
```

Warning: Most email servers today have strict limits on the size of the message, so it is recommended to keep the size of the attachments small. A maximum size of 2 MB is a good rule of thumb.

Multiple messages at once

To send multiple messages in one shot, you can use `create_message()` to create the messages and then use `deliver()` to send them. This is very useful when sending personalized emails for multiple recipients:

```
from email.headerregistry import Address


async def handler(ctx):
    messages = []
    for recipient in [Address('Some Person', 'some.person', 'company.com'),
                      Address('Other Person', 'other.person', 'company.com')]:
        message = ctx.mailer.create_message(
            subject='Hi there, %s!' % recipient.display_name,
            sender='Example Person <example@company.com>',
```

```
            to=recipient, plain_body='How are you doing, %s?' % recipient.display_
↪name)
        messages.append(message)

    await ctx.mailer.deliver(messages)
```

Handling errors

If there is an error, a `DeliveryError` will be raised. Its `message` attribute will contain the problematic `EmailMessage` instance if the error is specific to a single message:

```
async def handler(ctx):
    try:
        await ctx.mailer.create_and_deliver(
            subject='Hi there!', sender='Example Person <example@company.com>',
            to='recipient@company.com', plain_body='Greetings from Example!')
    except DeliveryError as e:
        print('Delivery to {} failed: {}'.format(e.message['To'], e.error))
```

CHAPTER 3

Version history

This library adheres to Semantic Versioning.

1.0.0

- Initial release
- API reference