

# plainMailFactory 1.3

Robert Reiz

- 84.6% code coverage with JUnit 4.4 and Clover2
- Templating system for e-mails
- Multi-client capability
- Internationalization
- Load balancing for e-mail server
- Fall back server definition
- Retry function
- Defining delay time between the retries
- Very small footprint, 29 kb.

- Retrospection
- Download & Configuration
- Templating
- Sending the mail
- Localization
- Fallback, Retry and Delay
- For testing
- Logging

# 1

## *Retrospection*

- Requirement: Application should send a E-Mail
- Solution 1: JavaMail from SUN

```
// Create a mail session
java.util.Properties props = new java.util.Properties();
props.put("mail.smtp.host", smtpHost);
props.put("mail.smtp.port", ""+smtpPort);
Session session = Session.getDefaultInstance(props, null);

// Construct the message
Message msg = new MimeMessage(session);
msg.setFrom(new InternetAddress(from));
msg.setRecipient(Message.RecipientType.TO, new InternetAddress(to));
msg.setSubject(subject);
msg.setText(content);

// Send the message
Transport.send(msg);
```

- The JavaMail API from SUN is not very pretty.

- Solution 2: Apache commons email

```
HtmlEmail email = new HtmlEmail();
email.setHostName(host);
email.setFrom(fromEmail, fromName);
if (!"".equals(authUser) && !"".equals(authPassword)){
    email.setAuthentication(authUser, authPassword);
}
email.addReplyTo(replyTo!=null?replyTo:fromEmail);
```

- The commons email API is much more easier. It is a wrapper for the mail API from SUN.

- You can start to build your email like this:

```
StringBuffer sb = new StringBuffer();  
sb.append("Dear");  
sb.append(firstname);  
sb.append(" ");  
sb.append(lastname);  
sb.append(",");
```

- But it is a very bad fashion. Better is a template like this:

```
:dear:
```

```
Your new password: :neuesPasswort:
```

```
best regards,  
FooBar
```

- Requirement: Multi-client capability
- Requirement: Internationalization

```
if (user.getMandant().equals("mandant1") ){
    if (locale.equals("de"){
        sendPasswordHtmlDE();
    } else {
        sendPasswordHtmlEN();
    }
} else {
    if (locale.equals("de"){
        sendPasswordPlainDE();
    } else {
        sendPasswordPlainEN();
    }
}
```



- The source code for reading and replacing the templates was in all applications similar.
- The source code for reading and replacing the templates was moved by “copy and paste” from app. to app.
- And so the ploinMailFactory was born.

- OpenSource
- Apache 2.0 License
- Hosted on SourceForge
- Download: <https://sourceforge.net/projects/ploinmailfactor/>
- Doku: <http://www.ploinMailFactory.org>

# 2

## *Configuration*

- Just download the JAR from:  
<https://sourceforge.net/projects/ploinmailfactor/files/>
- Add the JAR file to your classpath

The ploinMailFactory expects a file "mail.properties" located in your root source-folder, where your hibernate.properties and log4j.properties are placed, too. This is the central configuration file for the ploinMailFacotry.

```
mailDirectory=service/businesslogic/mail  
propFileName=mail.properties  
htmlExtension=.html  
plainExtension=.txt  
singleThread=true
```

## **propFileName=mail.properties**

The name for further configuration files (overriding / extending the main configuration)

## **mailDirectory=service/businesslogic/mail**

The path to the directory (now called "mail directory"), where the e-mail templates are placed

## **htmlExtension=.html**

The extension for the html templates

## **plainExtension=.txt**

The extension for the plain text templates

## **singleThread=true**

If this property is false, the e-mail sending process is happening in the current thread. If this property is true, the ploidMailFactory will start a new thread for sending the e-mail. This behavior is very useful if you have multiple E-Mail-Server and long time-outs.

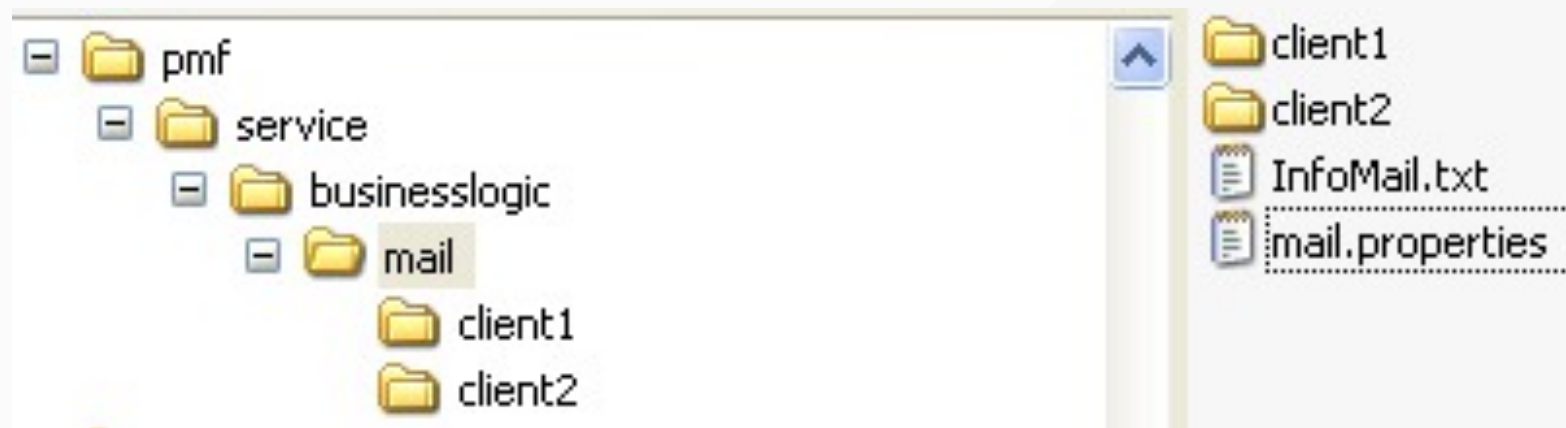
# 3

## *Templating*

The mail directory, where the e-mail templates are placed, should include a subdirectory for each client/mandant (now called "client directories") simply being named like the client. With 2 clients *client1* and *client2* we would have the following directory structure:

```
service/businesslogic/mail  
service/businesslogic/mail/client1  
service/businesslogic/mail/client2
```





- InfoMail.txt

:dear:

You are out of budget.

best regards,  
FooBar

- mail.properties

loadbalancing.nodes=server1

server1.host=your.email.server  
server1.fromEmail=foo@bar.org  
server1.fromName=Foo Bar  
server1.authUser=foo  
server1.authPassword=bar  
server1.replyTo=fo@bar.org

# 4

## *Sending the mail*

- The most important class in the ploinMailFactory is the MailFactory.

```
<bean id="mailFactory" class="org.ploin.pmf.impl.MailFactory" />

<bean id="postOffice" class="org.company.project.service.impl.PostOffice" >
  <property name="mailFactory" ref="mailFactory" />
</bean>
```

# Sending the mail

```
MailConfig mailConfig = new MailConfig();
mailConfig.addToRecipient("John Doe", "john@doe.com");
mailConfig.setSubject("InfoMail");

Map<String, String> map = new HashMap<String, String>();
map.put(":dear:", "Dear John Doe");

TemplateConfig templateConfig = new TemplateConfig();
templateConfig.setName("InfoMail");
templateConfig.setMap(map);

mailFactory.sendMail(mailConfig, templateConfig);
```

# 5

## *Localization*

- Just rename the template files.

```
InfoMail_en.html  
InfoMail_de.html
```

- Add the locale to the templateConfig.

```
TemplateConfig templateConfig = new TemplateConfig();  
templateConfig.setLocale(new Locale("de", "DE"));
```

- If the framework cannot find the template for the specified locale it uses the default (unlocalized) template.

# 6

## *Loadbalancing*

- Just add more servers to your config

```
loadbalancing.nodes=server1, server2
```

```
server1.host=your.email.server0  
server1.fromEmail=foo@bar.org  
server1.fromName=Foo Bar  
server1.authUser=foo  
server1.authPassword=bar  
server1.replyTo=fo@bar.org
```

```
server2.host=your.email.server1  
server2.fromEmail=foo@bar.org  
server2.fromName=Foo Bar  
server2.authUser=foo  
server2.authPassword=bar  
server2.replyTo=fo@bar.org
```



# 7

## *Fallback, Retry and Delay*

- Just add fallback servers to your config

```
fallback.nodes=fallback1, fallback2
```

```
fallback1.host=your.fallback.server1  
fallback1.fromEmail=foo@bar.org  
fallback1.fromName=Foo Bar  
fallback1.authUser=foo  
fallback1.authPassword=bar  
fallback1.replyTo=fo@bar.org  
fallback1.retry=3  
fallback1.delay=10000
```

```
fallback2.host=your.fallback.server2  
fallback2.fromEmail=foo@bar.org  
fallback2.fromName=Foo Bar  
fallback2.authUser=foo  
fallback2.authPassword=bar  
fallback2.replyTo=foo@bar.org  
fallback2.retry=3  
fallback2.delay=10000
```

- If you are working with fallback servers, it is useful to set the property  
"singleThread=false"

# 8

## *For Testing*

- For test scenarios it is useful to set the following properties in the mail.properties in the root.

```
toEmailOverride=test@mail.de  
toCcOverride=test@mail.de  
toBccOverride=test@mail.de
```

- If this properties are set, the list of "toRecipients", "ccRecipients" and "bccRecipients" in the mailConfig object will be replaced with the values from the properties file.

# 9

## *Logging*

- The ploinMailFactory uses intern log4j for logging. You can set the log level to debug with the following line in your log4j.properties.

```
log4j.logger.org.ploin.pmf DEBUG
```

- Or to ERROR

```
log4j.logger.org.ploin.pmf ERROR
```

???