Seam Mail

Seam Mail Introduction

Seam mail is an portable CDI extension designed to make working with Java Mail easier via standard methods or `plugable` templating engines.

1.1. Getting Started

No better way to start off then with a simple example to show what we are talking about.

```
@Inject
private Instance<MailMessage> mailMessage;

public void sendMail() {

   MailMessage m = mailMessage.get();
   m.from("John Doe<john@test.com>")
     .to("Jane Doe<jane@test.com>")
     .subject(subject)
     .bodyHtml(htmlBody)
     .importance(MessagePriority.HIGH)
     .send();
}
```

Very little is required to enable this level of functionality in your application. Let's start off with a little required configuration.

Configuration

By default the configuration parameters for Seam Mail are handled via configuration read from your application's seam-beans.xml. This file is then parsed by Seam Solder to configure the MailConfig class. You can override this and provide your own configuration outside of Seam Mail but we will get into that later.

2.1. Minimal Configuration

First lets add the relevant maven configuration to your pom.xml

```
 <dependency>
  <groupId>org.jboss.seam.mail</groupId>
  <artifactId>seam-mail-impl</artifactId>
  <version>${seam.mail.version}</version>
</dependency>
```

Now now that is out of the way lets provide `JavaMail` with the details of your `SMTP` server so that it can connect and send your mail on it's way.

This configuration is handled via Seam Solder which reads in the configuration from your application's `seam-beans.xml` and configures the `MailConfig` class prior to injection.

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:s="urn:java:ee"
       xmlns:mail="urn:java:org.jboss.seam.mail.core"
       xsi:schemaLocation="
      http://java.sun.com/xml/ns/javaee
      http://docs.jboss.org/cdi/beans_1_0.xsd">

    <mail:MailConfig
          serverHost="my-server.test.com"
          serverPort="25">
       <s:modifies/>
    </mail:MailConfig>

</beans>
```

That is all the configuration necessary to send a simple email message. Next we will take a look at how to configure and use the supported templating engines.

Important

JBoss AS 7.0.x does not correctly load all the modules to support sending mail *AS7-1375* [https://issues.jboss.org/browse/AS7-1375]. This is easily fixed By replacing the module definition at $JBOSS_HOME/modules/javax/activation/api/main/module.xml with the following

```
<module xmlns="urn:jboss:module:1.0" name="javax.activation.api">
   <dependencies>
      <module name="javax.api" />
      <module name="javax.mail.api" >
         <imports><include path="META-INF"/></imports>
      </module>
   </dependencies>

    <resources>
      <resource-root path="activation-1.1.1.jar"/>

      <!-- Insert resources here -->
   </resources>
</module>
```

This will be fixed in AS 7.1.x

Core Usage

3.1. Intro

While Seam Mail does provide methods to produce templated email, there is a core set of functionality that is shared whether you use a templating engine or not.

3.2. Contacts

At it's base an email consists of various destinations and content. Seam Mail provides a wide varerity of methods of ways to configure the following address fields

- From
- To
- CC
- BCC
- REPLY-TO

3.2.1. String Based

Seam Mail leverages the JavaMail InternetAddress object internally for parsing and storage and provides a varargs method for each of the contact types. Thus you can provide either a String, multiple Strings or a String []. Addresses are parsed as RFC 822 addresses and can be a valid Email Address or a Name + Email Address.

```
MailMessage m = mailMessage.get();
m.from("John Doe<john@test.com>")
.to("jane@test.com")
.cc("Dan<dan@test.com", "bill@test.com")
```

3.2.2. InternetAddress

Since we leverage standard InternetAddress object we might as well provide a method to use it.

```
MailMessage m = mailMessage.get();
m.from(new InternetAddress("John Doe<john@test.com>"))
```

3.2.3. EmailContact

Since applications frequently have their own object to represent a user who will have an email set to them we provide a simple interface which your object can implement.

```
public interface EmailContact {
  public String getName();

  public String getAddress();
}
```

Let's define this interface on an example user entity

```
@Entity
public class User implements EmailContact {

  private String username; //"john@test.com"
  private String firstName; //"John"
  private String lastName; //"Doe"

  public String getName() {
    return firstName + " " + lastName;
  }

  public String getAddress() {
    return username;
  }
}
```

Now we can use our User object directly in an of the contact methods

```
User user;

MailMessage m = mailMessage.get();
m.from("John Doe<john@test.com>")
.to(user)
```

3.2.4. Content

TODO

3.2.5. Attachments

TODO

Templating

4.1. Velocity

TO DO

4.2. Freemarker

TO DO

Advanced Features

5.1. MailTransporter

TO DO

5.2. MailConfig

TO DO