Apache Commons Email ™

Last Published: 21 February 2014 | Version: 1.4.0-SNAPSHOT

ApacheCon Apache Commons

EMAIL

- Overview
- Download
- User guide
- Latest API docs (development)
- Javadoc (1.3.2 release)
- Javadoc (1.2 release)

DEVELOPMENT

- Building

PROJECT DOCUMENTATION

- Project Information
- Project Reports

COMMONS

- Home
- License
- Components
- Sandbox
- Dormant

GENERAL INFORMATION

- Volunteering
- Contributing Patches
- Building Components
- Releasing Components
- Wiki

ASF

- How the ASF works
- Get Involved
- Developer Resources
- Sponsorship
- Thanks

APACHECON DENVER WESTIN DENVER DOWNTOWN APRIL 7-9, 2014

Built by: maven

A simple text email

Our first example will create a basic email message to "John Doe" and send it through your Google Mail (GMail) account.

```
Email email = new SimpleEmail();
email.setHostName("smtp.googlemail.com");
email.setSmtpPort(465);
email.setAuthenticator(new DefaultAuthenticator("username", "password"));
email.setSSLOnConnect(true);
email.setFrom("user@gmail.com");
email.setSubject("TestMail");
email.setMsg("This is a test mail ... :-)");
email.addTo("foo@bar.com");
email.send();
```

The call to setHostName("mail.myserver.com") sets the address of the outgoing SMTP server that will be used to send the message. If this is not set, the system property "mail.host" will be used.

Sending emails with attachments

To add attachments to an email, you will need to use the MultiPartEmail class. This class works just like SimpleEmail except that it adds several overloaded attach() methods to add attachments to the email. You can add an unlimited number of attachments either inline or attached. The attachments will be MIME encoded.

The simplest way to add the attachments is by using the EmailAttachment class to reference your attachments.

In the following example, we will create an attachment for a picture. We will then attach the picture to the email and send it.

```
import org.apache.commons.mail.*;
...

  // Create the attachment
  EmailAttachment attachment = new EmailAttachment();
  attachment.setPath("mypictures/john.jpg");
  attachment.setDisposition(EmailAttachment.ATTACHMENT);
  attachment.setDescription("Picture of John");
  attachment.setName("John");

  // Create the email message
  MultiPartEmail email = new MultiPartEmail();
  email.setHostName("mail.myserver.com");
  email.addTo("jdoe@somewhere.org", "John Doe");
  email.setFrom("me@apache.org", "Me");
  email.setSubject("The picture");
  email.setMsg("Here is the picture you wanted");

  // add the attachment
  email.attach(attachment);

  // send the email
  email.send();
```

You can also use EmailAttachment to reference any valid URL for files that you do not have locally. When the message is sent, the file will be downloaded and attached to the message automatically.

The next example shows how we could have sent the apache logo to John instead.

```
import org.apache.commons.mail.*;
...

  // Create the attachment
  EmailAttachment attachment = new EmailAttachment();
  attachment.setURL(new URL("http://www.apache.org/images/asf_logo_wide.gif"));
  attachment.setDisposition(EmailAttachment.ATTACHMENT);
  attachment.setDescription("Apache logo");
  attachment.setName("Apache logo");

  // Create the email message
  MultiPartEmail email = new MultiPartEmail();
  email.setHostName("mail.myserver.com");
  email.addTo("jdoe@somewhere.org", "John Doe");
  email.setFrom("me@apache.org", "Me");
  email.setSubject("The logo");
  email.setMsg("Here is Apache's logo");

  // add the attachment
  email.attach(attachment);

  // send the email
  email.send();
```

Sending HTML formatted email

Sending HTML formatted email is accomplished by using the HtmlEmail class. This class works exactly like the MultiPartEmail class with additional methods to set the html content, alternative text content if the recipient does not support HTML email, and add inline images.

In this example, we will send an email message with formatted HTML content with an inline image.

```
import org.apache.commons.mail.HtmlEmail;
...

  // Create the email message
  HtmlEmail email = new HtmlEmail();
  email.setHostName("mail.myserver.com");
  email.addTo("jdoe@somewhere.org", "John Doe");
  email.setFrom("me@apache.org", "Me");
  email.setSubject("Test email with inline image");

  // embed the image and get the content id
  URL url = new URL("http://www.apache.org/images/asf_
logo_wide.gif");
  String cid = email.embed(url, "Apache logo");

  // set the html message
  email.setHtmlMsg("<html>The apache logo - <img src=\
"cid:"+cid+"\"></html>");

  // set the alternative message
  email.setTextMsg("Your email client does not support
 HTML messages");

  // send the email
  email.send();
```

First, notice that the call to embed() returns a String. This String is a randomly generated identifier that must be used to reference the image in the image tag.

Next, there was no call to setMsg() in this example. The method is still available in HtmlEmail but it should not be used if you will be using inline images. Instead, the setHtmlMsg() and setTextMsg() methods were used.

Sending HTML formatted email with embedded images

The previous example showed how to create a HTML email with embedded images but you need to know all images upfront which is inconvenient when using a HTML email template. The ImageHtmlEmail helps you solving this problem by converting all external images to inline images.

```
import org.apache.commons.mail.HtmlEmail;
...

  // load your HTML email template
  String htmlEmailTemplate = ....

  // define you base URL to resolve relative resource
locations
  URL url = new URL("http://www.apache.org");

  // create the email message
  HtmlEmail email = new ImageHtmlEmail();
  email.setDataSourceResolver(new DataSourceResolverIm
pl(url));
  email.setHostName("mail.myserver.com");
  email.addTo("jdoe@somewhere.org", "John Doe");
  email.setFrom("me@apache.org", "Me");
  email.setSubject("Test email with inline image");

  // set the html message
  email.setHtmlMsg(htmlEmailTemplate);

  // set the alternative message
  email.setTextMsg("Your email client does not support
 HTML messages");

  // send the email
  email.send();
```

First we create a HTML email template referencing some images. All referenced images are automatically transformed to inline images starting from the current working directory.

Debugging

The JavaMail API supports a debugging option that will can be very useful if you run into problems. You can activate debugging on any of the mail classes by calling setDebug(true). The debugging output will be written to `System.out`.

Sometimes you want to experiment with various security setting or features of commons-email. A good starting point is the test class `EmailLiveTest` and `EmailConfiguration` which are used for testing commons-email with real SMTP servers.

Authentication

If you need to authenticate to your SMTP server, you can call the `setAuthentication(userName,password)` method before sending your email. This will create an instance of `DefaultAuthenticator` which will be used by the JavaMail API when the email is sent. Your server must support RFC2554 in order for this to work.

You can perform a more complex authentication method such as displaying a dialog box to the user by creating a subclass of the `javax.mail.Authenticator` object. You will need to override the `getPasswordAuthentication()` method where you will handle collecting the user's information. To make use of your new `Authenticator` class, use the `Email.setAuthenticator` method.

Security

Nowadays you should not use plain SMTP protocol when using public SMTP servers but there is a some confusion regarding the available options.

Two commons options are using

- STARTTLS on port 25
- SSL on port 465

The following definitions were taken from Wikipedia

- STARTTLS is an extension to plain text communication protocols, which offers a way to upgrade a plain text connection to an encrypted (TLS or SSL) connection instead of using a separate port for encrypted communication.
- Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide communication security over the Internet.TLS and SSL encrypt the segments of network connections above the Transport Layer, using asymmetric cryptography for key exchange, symmetric encryption

for privacy, and message authentication codes for message integrity.

In addition you can force the following security checks (which are disabled by default)

- When using a secured transport (STARTTLS or SSL) you can force validating the server's certificate by calling `Email.setSSLCheckServerIdentity(true)`.
- Enforce using STARTTLS by calling `Email.setStartTLSRequired(true)`

Handling Bounced Messages

Normally, messages which cannot be delivered to a recipient are returned to the sender (specified with the `from` property). However, in some cases, you'll want these to be sent to a different address. To do this, simply call the `setBounceAddress(emailAddressString)` method before sending your email.

Technical notes: When SMTP servers cannot deliver mail, they do not pay any attention to the contents of the message to determine where the error notification should be sent. Rather, they refer to the SMTP "envelope sender" value. JavaMail sets this value according to the value of the `mail.smtp.from` property on the JavaMail `Session`. (Commons Email initializes the JavaMail `Session` using `System.getProperties()`) If this property has not been set, then JavaMail uses the "from" address. If your email bean has the `bounceAddress` property set, then Commons Email uses it to set the value of `mail.smtp.from` when the `Session` is initialized, overriding any other value which might have been set.

Note: This is the only way to control the handling of bounced email. Specifically, the "Errors-to:" SMTP header is deprecated and cannot be trusted to control how a bounced message will be handled. Also note that it is considered bad practice to send email with an untrusted "from" address unless you also set the bounce address. If your application allows users to enter an address which is used as the "from" address on an email, you should be sure to set the bounce address to a known good address.