ERJavaMail Framework

Overview

ERJavaMail provides a simple and powerful API for sending component-based emails from a WebObjects application. ERJavaMail does not depend on any other pieces of Project Wonder, and is a good way to get your feet wet using Wonder.

Configuration

- er.javamail.centralize = true
  Centralize sends all emails to the er.javamail.adminEmail user.
- er.javamail.debugEnabled = true
  Determines whether or not email debugging is displayed. This contains protocol-level debug information.
- er.javamail.adminEmail = user@domain.com
  The email address of the admin user to send centralized emails to. This is a required property.
- er.javamail.smtpHost = smtp.domain.com
  The SMTP host name to use. If this isn't set, mail.smtp.host will be checked and ultimately WOHost will be used.
- er.javamail.senderQueue.size = 50
  The number of messages that the sender queue can hold. Defaults to 50.
- er.javamail.milliSecondsWaitIfSenderOverflowed = 6000
  The number of milliseconds to wait if the sender queue is full. Default is 6000.
- er.javamail.smtpAuth = true
  Sets whether or not Authenticated SMTP is used to send outgoing mail. If set, er.javamail.smtpUser MUST also be set (and preferably er.javamail.smtpPassword).
- er.javamail.smtpUser = smtpusername
  The username to use to login to the authenticated SMTP server.
- er.javamail.smtpPassword = smtppassword
  The password to use to login to the authenticated SMTP server.
- er.javamail.smtpProtocol = smtp
  The protocol to use to connect to the mail server. Defaults to smtp.
- er.javamail.XMailerHeader =
  The X-Mailer header to put into all outgoing mail messages. Defaults to nothing.
- er.javamail.defaultEncoding = UTF-8
  The default character encoding to use for message content. Defaults to ???.
- er.javamail.WhiteListEmailAddressPatterns =
  A comma-separated list of whitelisted email address patterns. If set, then only addresses that match one of the whitelisted patterns will delivered to. Pattern syntax is the same as EOQualifier's caseInsensitiveLike.
- er.javamail.BlackListEmailAddressPatterns =
  A comma-separated list of blacklisted email address patterns. If set, then any email addresses that match a blacklist pattern will not be delivered to. Pattern syntax is the same as EOQualifier's caseInsensitiveLike. The blacklist filter is processed last, so a blacklist pattern beats a whitelist pattern.
- er.javamail.smtpPort =
  To specify the TCP port number of your SMTP server. Default is 25, you can also try 587, who is the "submission" port.

You need to at least specify the value for er.javamail.centralize, and if the value is "true", you also have to specify a value for er.javamail.adminEmail.

Example Usage

```
// Create an instance of an ERMailDelivery subclass
 ERMailDeliveryHTML mail = new ERMailDeliveryHTML ();

 // Here ERMailDeliveryHTML needs a WOComponent to
 // render the HTML text content.
 mail.setComponent(mailPage);

 // Here you create a new instance of the message
 // You can loop over this fragment of code, not
 //  forgetting to use newMail ()
 // before you set the attributes of the message.
 try {
   mail.newMail();
   mail.setFromAddress(emailFrom);
   mail.setReplyToAddress(emailReplyTo);
   mail.setSubject(emailSubject);
   mail.setToAddresses(new NSArray (toEmailAddresses));
   // Send the mail.  There is an optional sendMail(boolean) that
   // optionally blocks during the send.
   mail.sendMail();
 } catch (Exception e) {
   // handle the exception ...
 }
```

Example of sending Mail with Attachments

The subclasses of ERMailDelivery will not only deliver the mail for you, but also will create the message. So ERMailDelivery is in some way more like a message than a delivery mechanism. For each message you want to sent instantiate a concrete subclass of ERMailDelivery (e.g. ERMailDeliveryPlainText). You can then add attachments to it. Below a simple code snippet sending of a mail with an attachment. The MimeType of the attachment is parsed out of the extension of the filename.

Sending mail is per default asynchronous, you can specify a flag (true h1. shouldBlock, false should not block) to influence the behaviour.

```
byte[] content; // assume this exist, same interface exists for NSData, too
 // Create an instance of an ERMailDelivery subclass
 ERMailDeliveryPlainText message = new ERMailDeliveryPlainText();
 // set the text and subject
 message.setTextContent("Some Mail text");
 message.setSubject("The mail subject");
 // add the attachment
 message.addAttachment(new ERMailDataAttachment("myattachment.zip", null,
content));
 message.setToAddress("receiver@bitbucket.com");
 message.setFromAddress("sender@bitbucket.com");
 // send the mail assynchronously
 message.sendMail();
```

Inline Attachments

If you have a WebObjects Component and you want to send inline images with the component, you have to make use of a trick, that was mentioned in Fabian Peters mail on the Wonder Discussion mailinglist on 6 june 2009. I could not find an online reference.

First of all, in the Component HTML one includes something like this:

```
<img src="cid:image0" alt="Logo" />
```

The code that accompanies this cid:image0 is:

```
URL logoURL = myApp().resourceManager().pathURLForResourceNamed("logo.png",
null, null);
File logo = new File(logoURL.toURI());
ERMailAttachment imageLogo = new
ERMailFileAttachment("logo.png","<image0>",logo);
```

Note that the file "logo.png" has to be included in the Resources folder of you WebObjects project.
The "cid:image0" is linked to the <image0> in the ImageAttachment.

Gotchas

Be careful of the WOContext that contains the component you are sending. If you use ERMailDeliveryHTML inside of the normal request-response loop with the default WOContext, it is very likely that the next page that is sent to the user will be the emailed component rather than the page you WANTED to send. There are several possible workarounds for this. One is to return a specific component rather than null from your action method. I have had better and more consistent success with the following code:

```
WOContext context = (WOContext) context().clone();
 MyComponent component = (MyComponent)
WOApplication.application().pageWithName(MyComponent.class.getName(),
context);
 ERMailDeliveryHTML mail = new ERMailDeliveryHTML();
 mail.setComponent(component);
 ...
```

This seems to properly isolate the email to a clone of the current context rather than the actual active context. Your mileage may vary.

If you are using Wonder 3.0 or later and you don't extend ERXApplication, you need to add this line in your Application constructor:

```
 ERJavaMail.sharedInstance ().finishInitialization();
```