Sending mail from applications in Java ME

Basics of Email

When you send an email to a friend, it looks as though the message goes directly from your mail program to your friend's. In reality, the message is handled by many servers that reside in the Internet infrastructure. Those servers talk to each other using standard protocols defined by groups such as IETF and W3C. For users, this process is entirely transparent. But developers have to understand exactly how an email message is transmitted to its destiny. In this section, we briefly go over important concepts and protocols in email.

The life cycle of an email message is as follows:

1. The sender's mail program contacts an SMTP (Simple Mail Transport Protocol) server. It tells the SMTP server the recipient and the content of the message.
2. The SMTP server finds the message recipient's mail server through an Internet registry and delivers the message to that server. The recipient server could be a POP3 (Post Office Protocol) or an IMAP (Internet Message Access Protocol) server.
3. The recipient POP3 or IMAP server stores the message. The next time the recipient logs onto that server, he/she will see that message.

SMTP Server

SMTP servers that do not require user authentication are called open relays. Open relays deliver all messages received. However, due to the abuse of spammers, most SMTP servers today require authentication. Your SMTP username and password are normally the same as the ones for your main ISP account.

POP3/IMAP Server

A POP3 server has only one inbox for each user. You should first download all email and then organize it using your mail client program. An IMAP server can provide multiple remote folders for each user. Those remote folders act as central message repositories and simplify email management when the user uses multiple mail client programs.

Introducing Mail4Me

Mail4ME is an Open Source library that provides lightweight email APIs for all J2ME profiles. Since Mail4ME is released under the CPL (Common Public License), you are free to integrate it into your proprietary software solutions. However, I do encourage you to release your changes and enhancements back to the community. This way, your additional features can be supported in future versions of Mail4ME. This book covers the v1.0 release of Mail4ME.

For devices that support TCP/IP socket connections, Mail4ME applications connect to Internet mail servers directly. But raw socket is not mandatory in MIDP specifications. For devices or networks that support only the mandatory HTTP, Mail4ME provides a special mode that utilizes an HTTP proxy. The source code of the proxy servlet and config instructions are included in the Mail4ME download package. As we will see, the use of the HTTP proxy is largely transparent to Mail4ME client developers.

Send Email

To send a text message via Mail4ME is very easy. You first need to compose a message and encapsulate it in a Message object. The listing below illustrates how to assemble a text message. All arguments passed to the Message constructor are in the String format.

Assemble an outgoing Message

```
Message message = new Message(my_address, recipient_address, subject);
message.addBodyLine( body );
```

Then, you connect the SMTP server and send the message via the SmtpClient object. The code comment illustrates the use of HTTP proxy. The message is successfully sent if the code does not throw an exception.

Send a Message using SmtpClient

```
SmtpClient smtpClient = null; // use proxy:
```

```
// smtpClient = new SmtpClient(
// new de.trantor.mail.http.ConnectionImpl(
// httpHost, 8080), hostname);
smtpClient = new SmtpClient(hostname);
smtpClient.open(smtpHost, 0, false, smtpUser, smtpPass);
smtpClient.sendMessage(message);
smtpClient.close();
```

Receive and Manipulate Messages

Now, let's see how we receive email via Mail4ME. First, we open a connection to the mail server via the InboxClient object. The code below demonstrates how to connect to POP3/IMAP servers with or without the HTTP proxy.

Connect to a POP3 Server

```
InboxClient inbox; if (imap) {
  // If you use a proxy
  // inbox = new ImapClient(
  // new de.trantor.mail.http.ConnectionImpl(
  // httpHost, 8080));
  inbox = new ImapClient();
} else {
  // If you use a proxy
  // inbox = new Pop3Client(
  // new de.trantor.mail.http.ConnectionImpl(
  // httpHost, 8080));
  inbox = new Pop3Client();
}
inbox.open(pop3Host, 0, false, pop3User, pop3Pass);
```

Now, we can get a list of new messages and their summaries. Method getMessageList() in the listing shown below returns an MIDP List object that displays a list of new emails. Each email's sender address and subject line is displayed as a List item. Vector msgNumbers keeps track of the index of each message in the list. This is necessary, since if you delete messages from the list, the list index goes out of sync with the message index on the POP3/IMAP server.

Retrieve Messages from the POP3 Server

```
private List getMessageList() throws MailException, IOException {   List result = new
List("Inbox", Choice.IMPLICIT);
  int count = inbox.getMessageCount();
  for (int i = 0; i < count; i++) {
    String uid = inbox.getUniqueId(i);
    int size = inbox.getSize(i);
    Message message = inbox.getHeaders(i);
    result.append(
      message.getHeaderValue("Subject", "No subject") + " ("
      + Message.getMachineAddress(
          message.getHeaderValue("From", "No sender"))
      + ")", null);
    // msgNumbers.insertElementAt(new Integer(i), 0);
    msgNumbers.addElement(new Integer(i));
  }
  return result;
}
```

The code shown below deletes the selected message.

Delete Messages from the POP3 Server

```
int num = msgList.getSelectedIndex();
inbox.removeMessage(((Integer)msgNumbers.elementAt(num)).intValue());
msgList.delete(msgList.getSelectedIndex());
msgNumbers.removeElementAt(num);
```

Display Message Parts

What if we want to display the details of a message? The following code illustrates how to display the selected message in an MIDP Form.

Displaying a Message

```
int num = msgList.getSelectedIndex(); if (num == -1) return;
Message message = inbox.getMessage(
  ((Integer)msgNumbers.elementAt(num)).intValue());
Form readScreen = new Form("Message details");
readScreen.append(
  new StringItem("From:",
    Message.getMachineAddress(
      message.getHeaderValue("From",
                             "No sender"))));
readScreen.append(
  new StringItem("Date:",
    message.getHeaderValue("Date", "No date")));
readScreen.append(
  new StringItem("Subject:",
    message.getHeaderValue("Subject",
      "No subject")));
MimeDecoder mime = new MimeDecoder(message);
addPartToScreen(mime, readScreen);
```

The method addPartToScreen() actually displays the content of the message based on the MIME types of its parts.

Display a multibody Message

```
private void addPartToScreen(MimeDecoder mime, Form screen) {   if (mime.getPartCount()
== 0) {
    if ("image/png".equals(mime.getType())) {
      byte[] bytes = mime.getBodyBytes();
      screen.append(Image.createImage(bytes, 0, bytes.length));
    } else if ((mime.getType() == null) ||
         ("text/plain".equals(mime.getType()))) {
      String s = "";
      for (int i = 0; i < mime.getBodyLineCount(); i++) {
        s = s + "\n" + mime.getBodyLine(i);
      }
      screen.append(s);
    } else {
      screen.append("\n[Unable to display \"" +
                    mime.getType() + "\" part.]");
    }
```

```
    } else {
      for (int p = 0; p < mime.getPartCount(); p++) {
        addPartToScreen(mime.getPart(p), screen);
      }
    }
  }
```

Reference

Note: This information was taken from the book Enterprise J2ME: Developing Mobile Java Applications. Please refer to that book for more information.