MAILBEANS DOCUMENTATION

by René Pirringer
MatNr.: 9656267
April 1999

Table of Contents

1 Overview

The MailBeans is a framework of JavaBeans which provide functionality for sending and receiving internet e-mail messages.
MailBeans supports following internet standards:

- POP3
- SMTP
- MIME

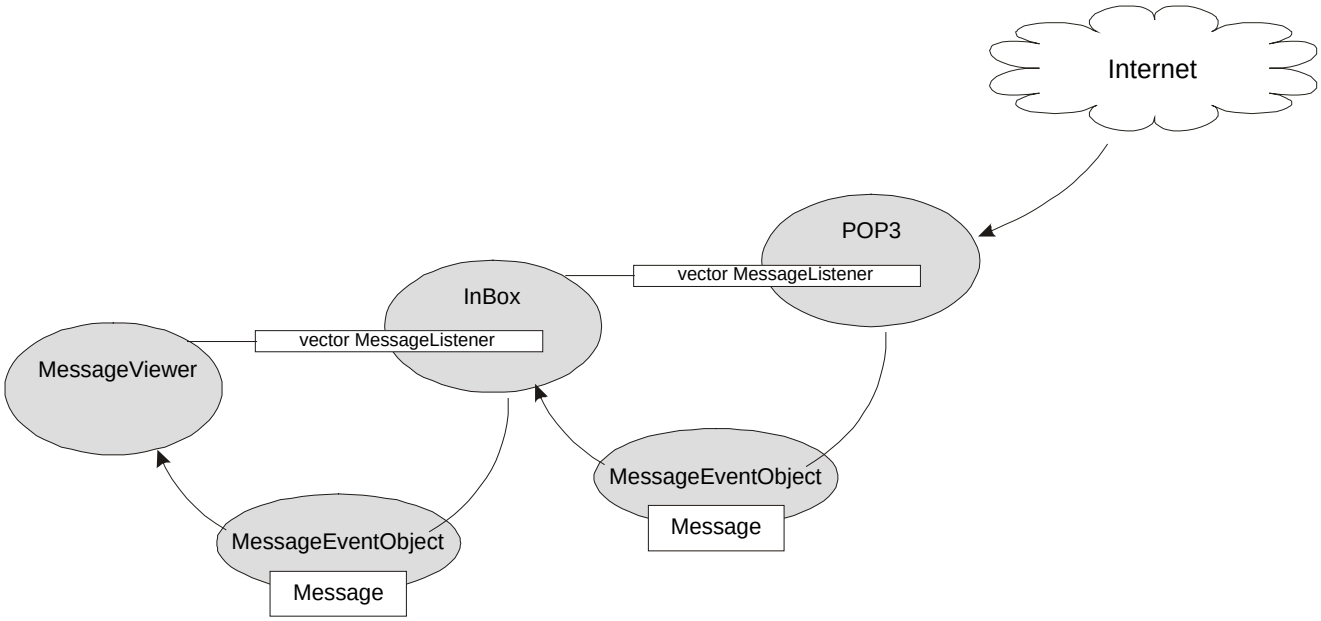
The MailBeans can be use for a mail-client written in Java, or also to add mail functionality to a existing application.

The framework consist of the followed beans:

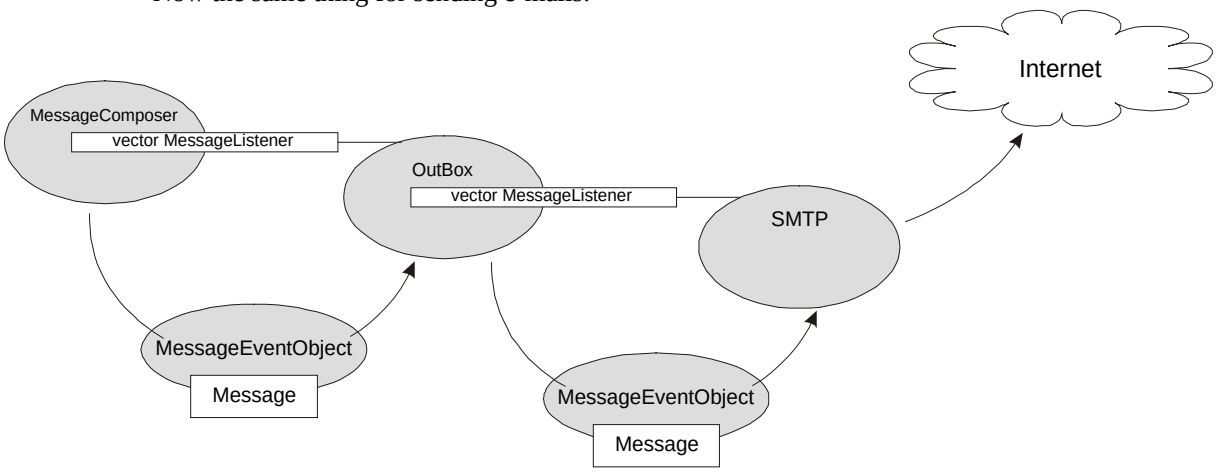
- POP3
- SMTP
- InBox
- OutBox
- MessageViewer
- MessageComposer
- SimpleMessageViewer
- SimpleMessageComposer
- OutputField
- SerializeData
- SettingsDialog

The beans communicate via MessageEvents. A MessageEvent contains an Message object, and this object contains the data. For every received e-mail an Message object must be created.

The pictures below shows how receiving e-mails work:



Now the same thing for sending e-mails:



The Message object can only handle simple text e-mails. For MIME e-mails the Message object must be replaced by an MimeMessage object.

2 Communication between the beans

There are three types of events I have implement:

- MessageEvent
- OutputEvent
- SerializePropertiesEvent

2.1 MessageEvent

The MessageEvent is used to transport a Message from one bean to another. For example: The POP3 bean receives a e-mail and creates a Message or a MimeMessage. This Message is bound to a MessageEvent that the POP3 bean fires. The InBox is a MessageEventListener and receives the MessageEvent and can add the Message.

2.1.1 Beans that can fire a MessageEvent:

- POP3: For every received e-mail the POP3 bean fires a MessageEvent.
- InBox: To show the a Message with a MessageViewer the InBox fires a MessageEvent.
- OutBox: For sending Message and to show queued Messages the OutBox fires a MessageEvent
- MessageComposer: To add the composed Message to the OutBox.

2.1.2 Beans that receives MessageEvent:

- SMPT: To get Messages from the OutBox or from a MessageComposer.
- OutBox: To get Messages from a MessageComposer.
- InBox: To get Messages from the POP3 bean.

2.1.3 How to receive MessageEvents:

To write a class that can receive MessageEvents the first thing is to implement the MessageListener interface. This interface consists of two methods:

- handleMessage(MessageEvent): to receive Messages from other classes (e.g. POP3 bean)
- showMessage(MessageEvent): for a MessageViewer to display the received Message

To get the Message from the MessageEvent the MessageEvent has the method getMessage() which returns the Message.

2.1.4 How to fire a MessageEvent:

To fire MessageEvents the first thing is to implement the functionality that MessageListener can register to receive MessageEvents.

Here is the source-code which must been added:

```
Vector messageListener = new Vector();

public void addMessageListener(MessageListener l) {
   messageListener.addElement(l);
}

public void removeMessageListener(MessageListener l) {
   messageListener.removeElement(l);
}
```

Here is the code to send a MessageEvents to registered listeners:

```
public void fireHandleMessage(Message msg) {
   MessageEvent me = new MessageEvent(this, msg);
   for (int i=0; i<messageListener.size(); i++) {
      MessageListener ml = (MessageListener) messageListener.elementAt(i);
      ml.handleMessage(me);
   }
}

public void fireShowMessage(Message msg) {
   MessageEvent me = new MessageEvent(this, msg);
   for (int i=0; i<messageListener.size(); i++) {
      MessageListener ml = (MessageListener) messageListener.elementAt(i);
      ml.showMessage(me);
   }
}
```

2.2 OutputEvent

The OutputEvent is used to display information. For Example: The POP3 bean firesOutputEvent if the connection to the server is successfully or if it fails, or how many mail are on the server and will be downloaded.

2.2.1 Bean that fires OutputEvents:

- POP3
- SMTP
- InBox
- OutBox

2.2.2 Beans that receives OutputEvents

- OutputField
- SerializeProperties

2.2.3 How to receive OutputEvent:

To receive OutputEvents the OutputListener interface must be implemented. This interface consists of one method:

- handleOutput(OutputEvent)

The method getText() from the OutputEvent returs a string with the information.

Example:

```
public class OutputField extends java.awt.TextField implements OutputListener {

       public OutputField() {
      super();
       }

       public void handleOutput(OutputEvent e) {
      if (e!=null) {
         setText(e.getText());
      }
       }
}
```

2.2.4 How to fire OutputEvents:

Code for registering listeners:

```
Vector outputListener = new Vector();

public void addOutputListener(OutputListener l) {
   outputListener.addElement(l);
}

public void removeOutputListener(OutputListener l) {
```

```
   outputListener.removeElement(l);
}
```

Code to send OutputEvents to registered listeners:

```
public void fireOutputEvent(String text) {
   OutputEvent e = new OutputEvent(this, text);
   for (int i=0; i<outputListener.size(); i++) {
      OutputListener ml = (OutputListener) outputListener.elementAt(i);
      ml.handleOutput(e);
   }
}
```

2.3 SerializeDataEvent

The SerializeDataEvent is used to serialize data from the beans (e.g. values from properties) in a single file.

2.3.1 Beans that receives SerializeDataEvent:

- InBox
- Outbox
- SettingDialog
- MessageViewer

2.3.2 Bean that fires a SerializeDataEvent:

- SerializeData

2.3.3 How to receive SerializeDataEvents:

To receive SerializeDataEvents the SerializeDataListener interface must be implemented. This interface consists of two methods:

- readData(SerializeDataEvent e)
- writeData(SerializeDataEvent e)

Examples:

```
public void readData(SerializeDataEvent e) {
   ObjectInputStream in = e.getObjectInputStream();
   try {
      obj = in.readObject();
   } catch (IOException ex) {
      System.out.println("IOException in mailbeans.MessageViewer.readProperties():
"+ex.getMessage());
   }
}

public void writeData(SerializeDataEvent e) {
   ObjectOutputStream out = e.getObjectOutputStream();
   try {
      out.writeObject(obj);
      out.writeObject(popServer);
      out.writeObject(popUsername);
      out.writeObject(popPassword);
      out.writeObject(smtpServer);
   } catch (IOException ex) {
      System.out.println("IOException mailbeans.SettingsDialog.writeProperties():
"+ex.getMessage());
   }
}
```

2.3.4 How to fire SerializeDataEvents:

Code for registering listeners:

```
Vector serializeDataListener = new Vector();

public void addSerializeDataListener (serializeDataListener l) {
   serializeDataListener.addElement(l);
}

public void removeSerializeDataListener(SerializeDataListener l) {
   serializeDataListener.removeElement(l);
}
```

Code to send SerializeDataEvents to registered listeners:

```
public void fireReadData() {
   try {
      FileInputStream in = new FileInputStream(filename);
      ObjectInputStream s = new ObjectInputStream(in);

      SerializeDataEvent e = new SerializeDataEvent(this, s);
      for (int i=0; i<serializeDataListener.size(); i++) {
         SerializeDataListener sdl = (SerializeDataListener)
serializeDataListener.elementAt(i);
         sdl.readData(e);
      }

      s.close();
   } catch (IOException ex) {
      System.out.println("IOException: "+ex.getMessage());
   }
}

public void fireWriteData() {
   try {
      FileOutputStream out = new FileOutputStream(filename);
      ObjectOutputStream s = new ObjectOutputStream(out);

      SerializeDataEvent e = new SerializeDataEvent(this, s);
      for (int i=0; i<serializeDataListener.size(); i++) {
         SerializeDataListener sdl = (SerializeDataListener)
serializeDataListener.elementAt(i);
         sdl.writeData(e);
      }
      s.close();
   } catch (IOException ex) {
      System.out.println("IOException: "+ex.getMessage());
   }
}
```

3 MailBeans Tutorial

This collection of JavaBeans includes beans to handle e-mail communication over internet. This beans implements the POP3 protocol for receiving messages and the SMTP protocol for sending messages. This messages can be simple text-messages or MIME-messages. How to use this is described in this document with examples.

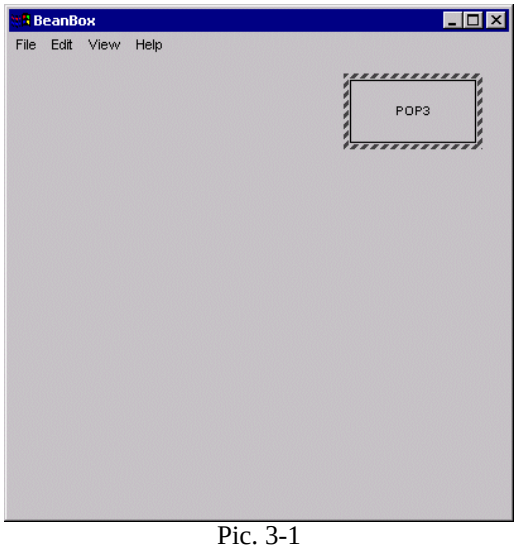
All the samples are tested with the BeanBox in Suns BDK1.0, but this beans can only run in an application, not as applet.

3.1 Simple POP3 Client.

The first example is a simple mail client with the POP3 bean.
For this program the followed beans are used:

- POP3
- OurButton (from the BDK1.0)
- InBox
- SimpleMessageViewer

The first step is to start the BeanBox. If the MailBeans.jar file is in the /bdk/jars directory the required beans are automatically loaded. Now you can select the POP3 bean in theToolBox an place it in the BeanBox. This looks like Pic 1-1.



Pic. 3-1

The next thing to do is to set the properties:

Server settings:

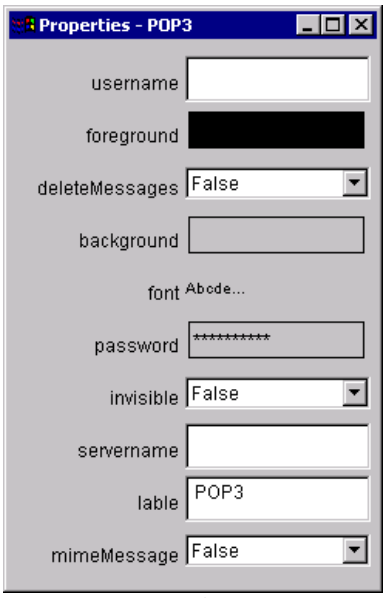
- servername: servername of the POP3 server you want to connect (always required)
- username: username for the POP3 server (always required)
- password: password for the POP3 server (always required)

Visual settings:

- background: backgroundcolor of the bean
- foreground: foregroundcolor of the bean
- label: text that displayes the bean
- font: font of the text
- invisible: hides the bean

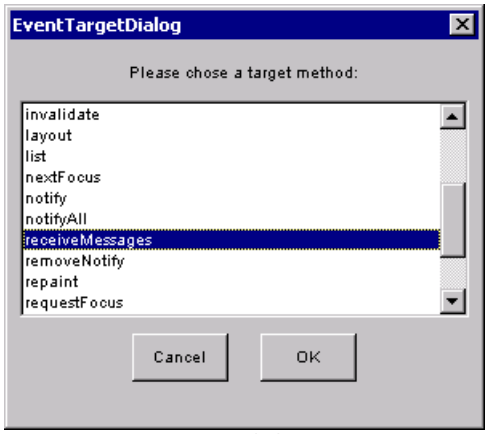
Advanced settings:

- deleteMessages: deletes the messages on the server after receiving
- mimeMessage: the bean creates MimeMessages if this is true. This is used if you work with a MimeMessage-Viewer



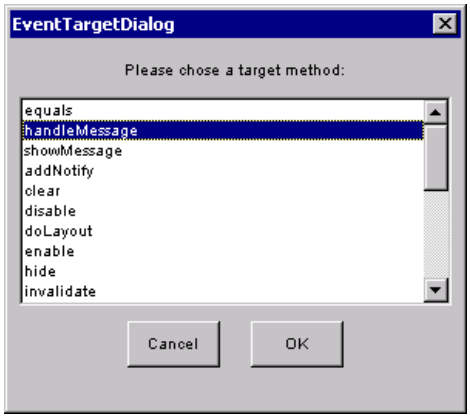
Pic. 3-2

The next step is to add a Button and connect it with the POP3 bean. I use theOurButton in the example, but you can also use any kind of button. Add the button to theBeanBox, label it “Get Mails” and select it. Select the Edit -> Events –> action ->actionPerformed pulldown menu. Connect the button now with the POP3 bean. Now the EventTargentDialog is shown (Pic 1-3). Select the receiveMessages method and press OK.



Pic. 3-3

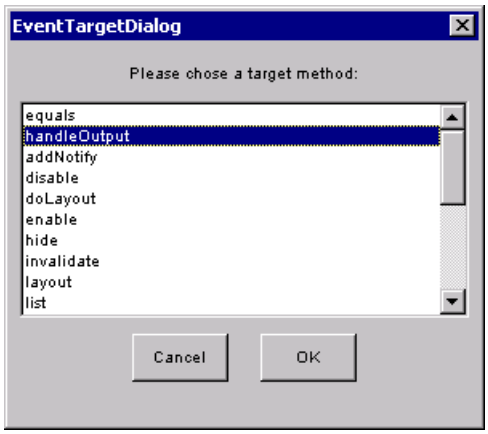
The next you must add is a bean that can show the received messages. I use here theInBox bean. Add it to the BeanBox.
To connect the SimpleInBox with the POP3 beans you must select theInBox. Than select the Edit-> Events -> message -> handleMessage pulldown menu and connect now to the POP3 bean.
Now the EventTargetDialog is shown. Select the handleMessage method and press OK (Pic 1-4).



Pic. 3-4

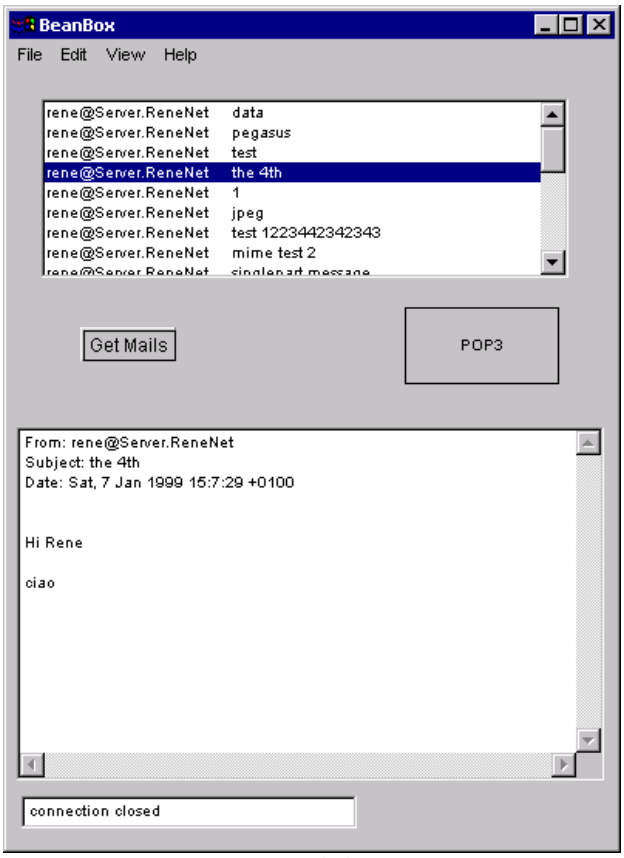
To view the mails you received from the POP3 server we must add aMessageViewer. I use the SimpleMessageViewer in this example. Place it in the BeanBox. To connect the InBox with the SimpleMessageViewer select the SimpleInBox then select the Edit-> Events -> message -> handleMessage pulldown menu and connect it to the SimpleMessageViewer. The EventTargetDialog is shown, select the handleMessage method and press OK.

The last thing in this example is the OutputField. It is not needed, but it is very useful, because it will display messages of the status of the connection, of display errors.
Add the OutputField in the BeanBox. Select the POP3 bean and select the Edit -> Events -> output -> handleOutput pulldown menu. Connect it to the OutputField and select from the EventTargetDialog the handleOutput method (Pic 1-5).



Pic. 3-5

To test the Beans press the “Get Mails” Button. Now your mails were downloaded from the POP3 server. They are displayed in the SimpleInBox. If you make a double-click on a mail, the mail is displayed in the SimpleMessageViewer (Pic 1-6).



Pic. 3-6

3.2 Simple SMTP Client

This example should demonstrate how the SMTP bean works.
For this program the followed beans are used:

- SMTP
- SimpleMessageComposer
- OutputField

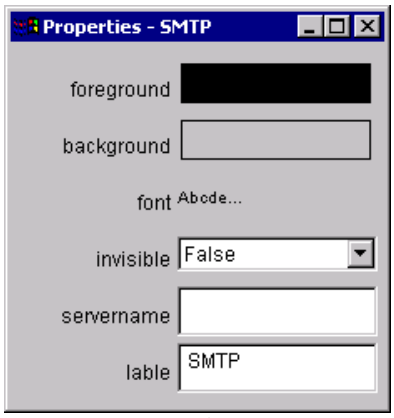
First select the SMTP bean and add in into the BeanBox. This Bean looks like the POP3 bean in the first example. The next thing is to do is to set the properties (Pic 2-1).

Server setting:

- servername: Enter the servername for the SMTP server you want to use. (always requierd)

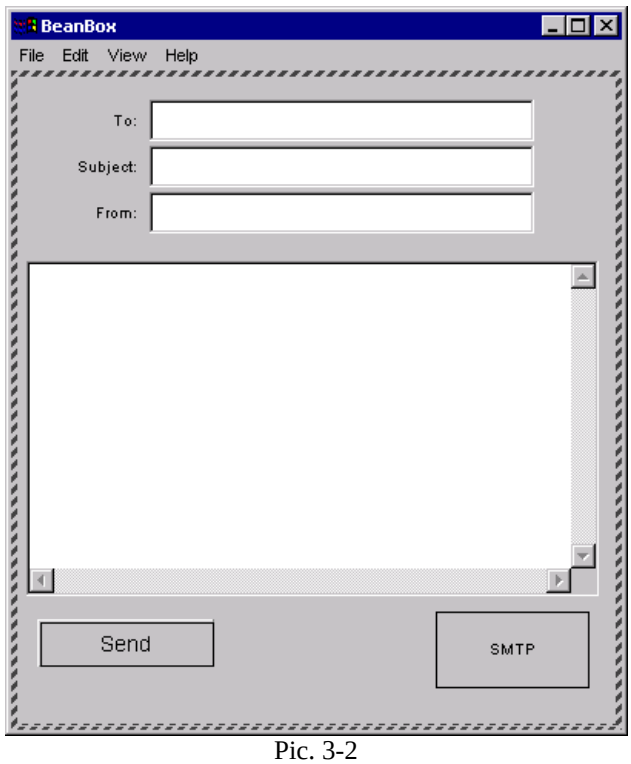
Visual Settings:

- background: backgroundcolor of the bean
- foreground: foregroundcolor of the bean
- label: text that displayes the bean
- font: font of the text
- invisible: hides the bean



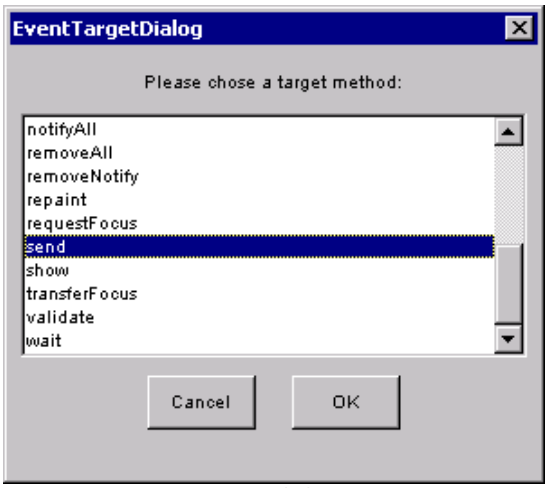
Pic. 3-1

The next thing is to add the SimpleMessageComposer, and a Button like the OurButton from the BDK. Rename the button with “Send” (Pic. 2-2).



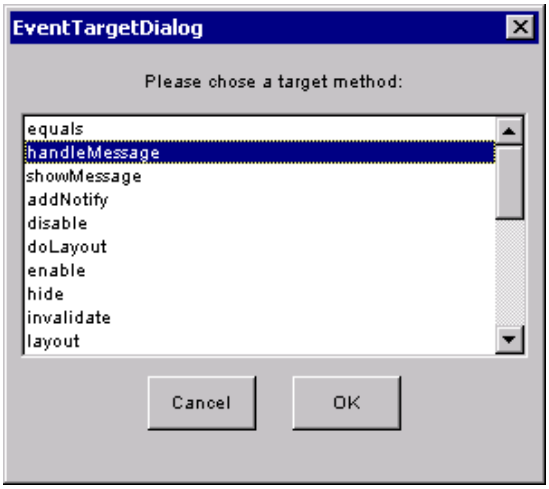
Pic. 3-2

Now the beans must be connected together. After selecting the Send-button select Edit->Events->action->actionPerformed. The EventTargedDialog is shown (Pic 2-3). Mark the send methode and press OK. The send method generates an MessageEvent with a Message that is generated from the data you entered in the SimpleMessageComposer.



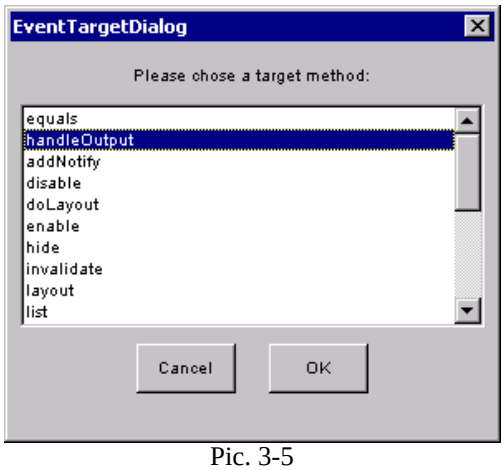
Pic. 3-3

The SMTP bean should receive this event, so the SimpleMessageComposer must be connected to the SMTP bean. Select the SimpleMessageComposer and connect it with Edit->Events->message->handleMessage and the methode handleMessage from the TargetEventDialog (Pic 2-4).



Pic. 3-4

The last thing in this example is the OutputField. It is not needed, but it is very useful, because it will display messages of the status of the connection, of display errors.
Add the OutputField in the BeanBox. Select the SMTP bean and select the Edit -> Events -> output -> handleOutput pulldown menu. Connect it to the OutputField and select from the EventTargetDialog the handleOutput method (Pic 2-5).



Pic. 3-5

That's all. Fill out the form and send the mail by pressing the Send button. The from is cleared after sending automatically.

3.3 How to write a MessageViewer

To write a MessageViewer is very simple. The only thing to do is to implement the MessageListener interface. This interface consists of one method which is called "handleMessage(MessageEvent e)", and it is called when a MessageEvent occurs. A MessageEvent contained a Message. The getMessage() method of the MessageEvent returns this Message.

Here is the sourcecode of SimpleMessageViewer the is used in the Simple POP3 Client example.

```
package mailbeans;

import java.awt.*;
import java.util.*;
/**
 * A very simple MessageViewer example
 */
public class SimpleMessageViewer extends TextArea implements MessageListener {
   Frame f;
   TextArea ta;

public SimpleMessageViewer() {
}
public void handleMessage(MessageEvent e) {
   Message message = e.getMessage();
   StringBuffer sb = new StringBuffer();
   sb.append("From: "+message.getFrom()+"\n");
   sb.append("Subject: "+message.getSubject()+"\n");
   sb.append("Date: "+message.getDate()+"\n\n");
   sb.append(message.getMsgBodyString());
   setText(sb.toString());
}
}
```

3.4 How to write a InBox

To write a InBox is also simple, but you need a bit more of code. There is the classMailBox with methods for adding, deleting, storing and loading Messages.
Here is the source code of the InBox beans, which implements the look and feel of the bean, and the rest is done in the MailBox class.

```
package mailbeans;

import java.awt.*;
import java.awt.event.*;
```

```
import java.util.*;

public class InBox extends MailBox implements ActionListener {
   java.awt.List list;

   public InBox() {
      super();
      setLayout(new BorderLayout());
      list = new java.awt.List();
      add(list);
      list.addActionListener(this);
   }

   public void actionPerformed(ActionEvent e) {
      if (e!=null) {
         fireShowMessage(getSelectedMessage());
      }
   }

   /* This method is called from the handleMessage method from the superclass
   public void addMessage(Message msg) {
      super.addMessage(msg);
      String from ="";
      StringTokenizer st = new StringTokenizer(msg.getFrom(), "<>");
      while (st.hasMoreTokens()) {
         from=st.nextToken();
      }
      list.add(from+"     "+msg.getSubject());
   }

   public void deleteMessage(int pos) {
      super.deleteMessage(pos);
      list.remove(0);
   }

   public Dimension getPreferredSize() {
      return new Dimension(100,100);
   }

   public Message getSelectedMessage() {
      return getMessage(list.getSelectedIndex());
   }

   public void handleMessage(MessageEvent e) {
      super.handleMessage(e);
   }
}
```

3.5 How to write a OutBox

The Outbox is the same like the InBox, but the thing that is different is that the OutBox must fire a handleMessage MessageEvent for sending Messages.

Add this method and it should work:

```
public void sendMail() {
   while(list.getItemCount()>0) {
      fireHandleMessage(getMessage(0));
      deleteMessage(0);
   }
}
```

3.6 How to write a MessageComposer

This a longer source code, but the most important part is the send() method. In this method the Message object is created an a MessageEvent is fired with this Message.

```
package mailbeans;

import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class SimpleMessageComposer extends Panel {
   private TextArea taMessage;
   private Label lTo;
   private Label lSubject;
   private Label lFrom;
   private TextField tfTo;
   private TextField tfSubject;
   private TextField tfFrom;
   Vector messageListener = new Vector();

   public SimpleMessageComposer() {
      super();
      setLayout(null);
      lTo = new Label("To: ");
      lTo.setAlignment(Label.RIGHT);
      add(lTo);
      tfTo = new TextField();
      add(tfTo);

      lSubject = new Label("Subject: ");
      lSubject.setAlignment(Label.RIGHT);
      add(lSubject);
      tfSubject = new TextField();
      add(tfSubject);

      lFrom = new Label("From: ");
      lFrom.setAlignment(Label.RIGHT);
      add(lFrom);
      tfFrom = new TextField();
      add(tfFrom);

      taMessage = new TextArea();
      add(taMessage);
   }

   public void addMessageListener(MessageListener l) {
      messageListener.addElement(l);
   }

   public void doLayout() {
      lTo.setBounds(5, 5, 70, 27);
      tfTo.setBounds(80, 5, 250, 27);
      lSubject.setBounds(5, 35, 70, 27);
      tfSubject.setBounds(80, 35, 250, 27);
      lFrom.setBounds(5, 65, 70, 27);
      tfFrom.setBounds(80, 65, 250, 27);
      taMessage.setBounds(0, 110, getSize().width, getSize().height-110);
   }

   public Dimension GetPreferredSize() {
      return new Dimension(300,300);
   }

   public void removeMessageListener(MessageListener l) {
      messageListener.removeElement(l);
   }

   public void send() {
      Message msg = new Message();
      msg.setTo(tfTo.getText());
      msg.setSubject(tfSubject.getText());
      msg.setFrom(tfFrom.getText());
      msg.setReturnPath(tfFrom.getText());
      msg.setMsgBody(taMessage.getText());
```

```
            msg.create(); // very important

            // Send Mail
            MessageEvent me = new MessageEvent(this, msg);
            for (int i=0; i<messageListener.size(); i++) {
               MessageListener ml = (MessageListener) messageListener.elementAt(i);
               ml.handleMessage(me);
            }
            tfSubject.setText("");
            tfTo.setText("");
            tfFrom.setText("");
            taMessage.setText("");
         }
      }
```

4 Mailbean Specification

4.1 POP3 Bean

extends MailBeansComponent

The POP3 bean provides functionality to receive e-mails from a POP3 server.

4.1.1 Properties:

Server settings:

- *servername:* servername of the POP3 server you want to connect
- *username:* username for the POP3 server
- *password:* password for the POP3 server

Visual settings:

- *background:* backgroundcolor of the bean
- *foreground:* foregroundcolor of the bean
- *label:* text that displayes the bean
- *font:* font of the text
- *invisible:* hides the bean

Advanced settings:

- *deleteMessages:* deletes the messages on the server after receiving
- *mimeMessage:* the receiveMessages methods will create MessageEvents with MimeMessages.

4.1.2 Methods:

- *receiveMessages():* Connect and log on to a POP3 server, and downloads all messages. This methods fires a MessageEvent for each downloaded message.

4.1.3 Events:

The POP3 bean fires:

- *OutputEvent*
- *MessageEvent*

4.2 SMTP Bean

extends MailBeanComponent
implements MessageListener

The SMTP bean provides functionality for sent e-mail.

4.2.1 Properties:

Server setting:

- servername: Enter the servername for the SMTP server you want to use.

Visual Settings:

- *background:* backgroundcolor of the bean
- *foreground:* foregroundcolor of the bean
- *label:* text that displays the bean
- *font:* font of the text
- *invisible:* hides the bean

4.2.2 Methods:

MessageListener methods:

- *handleMessage(mailbeans.MessageEvent):* receives a MessageEvent and sends the message from the MessageEvent
- *showMessage(mailbeans.MessageEvent):* this method is empty

4.2.3 Events:

The SMTP bean fires:

- *OutputEvent*

and receives:

- *MessageEvent*

4.3 MailBox

is abstract
extends MailBeansContainer
implements MessageListener, SerializeDataListener

The MailBox beans is a abstract class to provide functionality for handling, storing and loading Messages. This class is the superclass of the InBox and the OutBox.

4.3.1 Properties:

- *path:* The path of the director the Messages were stored.

4.3.2 Methods:

- *addMessage(mailbeans.Message):*
  This method is called by the handleMessage(mailbeans.MessageEvent) method and add this Message.
- *deleteMessage(int):*
  Deletes the Message on specified position.
- *fireHandleMessage(mailbeans.Message):*
  Fires a handleMessage MessageEvent and includes the specified Message.
- *fireShowMessage(mailbeans.Message):*
  Fires a showMessage MessageEvent and includes the specified Message.
- *mailbeans.Message getMessage(int index):*
  Gets the specified Message.
- *String getPath():* Gets the path of the directory were the mails are stored.
- *output(String text):*
  Fires a OutputEvent with the specified text.
- *setPath(String path):*
  Gets the path of the directory were the mails are stored.

MessageListener methods:

- *handleMessage(mailbeans.MessageEvent):*
  receives a MessageEvent and add this Message with the method addMessage(mailbeans.Message) to the Mailbox.
- *showMessage(mailbeans.MessageEvent):*
  This method is empty.

SerializeDataListener:

- *readDate(mailbeans.SerializeDataEvent):*
  Reads the Settings from the InputStream which the SerializeDataEvent consist.
- *writeData(mailbeans.SerializeDataEvent):*
  Writes the Settings from the InputStream which the SerializeDataEvent consist.

4.3.3 Events:

The InBox bean fires:

- *MessageEvent*
- *OutputEvent*

and receives:

- *MessageEvent*

4.4 InBox

extends MailBox
implements ActionListener

The InBox beans receives Messages via MessageEvents and stores this. The functionality for handling, storing and loading Messages is implemented in the MailBox class. In this class only the look and feel of the InBox is implemented.

4.4.1 Properties:

See MailBox.

4.4.2 Methods:

MessageListener methods:

- *handleMessage*(mailbeans.MessageEvent)*:*
  receives a MessageEvent and add this Message into the Inbox

4.4.3 Events:

The InBox bean fires:

- *MessageEvent*
- *OutputEvent*

and receives:

- *MessageEvent*

4.5 OutBox

extends MailBox
implements ActionListener

The OutBox beans receives Messages via MessageEvents and stores this. The functionality for handling, storing and loading Messages is implemented in the MailBox class. In this class only the look and feel of the OutBox is implemented.

4.5.1 Properties:

See MailBox.

4.5.2 Methods:

- *sendMail():* Fires for each Message a MessageEvent. The listener for the events is the SMTP bean.

MessageListener methods:

- handleMessage(mailbeans.MessageEvent)
  receives a MessageEvent and add this Message into the Outbox

4.5.3 Events:

The OutBox bean fires:

- *MessageEvent*
- *OutputEvent*

and receives:

- *MessageEvent*

4.6 MessageViewer

extends Componet
implements WindowListener, ActionListener, ComponentListener, MessageListener, SerializeDataListener

The MessageViewer bean is used to view MimeMessage. The Message is display in a own Frame. The different MimeTypes uses different viewers.

4.6.1 Properties:

Visual Settings:

- *background:* backgroundcolor of the bean
- *foreground:* foregroundcolor of the bean
- *label:* text that displays the bean
- *font:* font of the text
- *invisible:* hides the bean

4.6.2 Methods:

MessageListener methods:

- *handleMessage(mailbeans.MessageEvent)*
  This method is empty.
- *showMessage(mailbeans.MessageEvent)*
  Receives a MessageEvent and displays this Message.

SerializeDataListener:

- *readDate(mailbeans.SerializeDataEvent)*
  Reads the Settings from the InputStream which the SerializeDataEvent consist.
- *writeData(mailbeans.SerializeDataEvent)*
  Writes the Settings from the InputStream which the SerializeDataEvent consist.

4.6.3 Events:

The MessageViewer bean receives:

- *MessageEvent*
- *SerializeDataEvent*

4.7 SimpleMessageViewer

extends TextArea
implements MessageListener

The SimpleMessageViewer bean is used to view Messages.

4.7.1 Properties:

see TextArea in the JDK documentation

4.7.2 Methods:

MessageListener methods:

- *handleMessage(mailbeans.MessageEvent)*
  This method is empty.
- *showMessage(mailbeans.MessageEvent)*
  Receives a MessageEvent and displays this Message.

4.7.3 Events:

The MessageViewer bean receives:

- *MessageEvent*

4.8 MessageComposer

extends MailBeansComponent
implements WindowListener, ActionListener, ComponentListener

The MessageComposer bean is used to compose a Message. A new Frame is opened to compose a Message.

4.8.1 Properties:

Visual Settings:

- *background:* backgroundcolor of the bean
- *foreground:* foregroundcolor of the bean
- *label:* text that displays the bean
- *font:* font of the text
- *invisible:* hides the bean

4.8.2 Methods:

- *compose():* Opens the MessageComposer frame.

4.8.3 Events:

The MessageComposer bean fires:

- *MessageEvent*

4.9 SimpleMessageComposer

extends Panel

The SimpleMessageComposer bean is used to compose a Message.

4.9.1 Methods:

- *send():* Fires a MessageEvent with the new Message which is created.

4.9.2 Events:

The MessageComposer bean fires:

- *MessageEvent*

4.10 OutputField

extends TextField

The OutputField is a TextField that can display the receiveOutputEvents an can display the information the OutputEvent consists of.

4.10.1 Methods:

OutputListener methods:

- *handleOutput(mailbeans.OutputEvent)*
  Receives a OutputEvent and displays the information.

4.10.2 Events:

The MessageComposer bean receives:

- *OutputEvent*

4.11 SettingsDialog

extends MenuItem
implements ActionListener, WindowListener, SerializeDataListener

The SettingsDialog is a Dialog to set up the server settings properties for the POP3 and the SMTP bean. This bean is a MenuItem, and it opens automatically the Dialog after choosing this item.

4.11.1 Properties:

- *email:* The email address of the user
- *popServer:* The name of the POP3 server
- *popUsername:* The username for the POP3 account
- *popPassword:* The password for the POP3 account
- *smptServer:* The name of the SMTP server

4.11.2 Methods:

SerializeDataListener:

- *readDate(mailbeans.SerializeDataEvent):*
  Reads the Settings from the InputStream which the SerializeDataEvent consist.
- *writeData(mailbeans.SerializeDataEvent):*
  Writes the Settings from the InputStream which the SerializeDataEvent consist.

4.11.3 Events:

The SettingsDialog bean fires:

- *ProperrtyChangeEvent*

and receives:

- *SerializeDataEvent*

4.12 SerializeData

extends MenuItem
implements ActionListener, WindowListener, SerializeDataListener

The SerializeData bean provides functionality to save settings (e.g. values of properties from beans). While loading and storing a dialog is opened to displayOutputEvents.

4.12.1 Properties:

- *filename*: *Name of the file were the settings were stored.*

Visual settings:

- *background:* backgroundcolor of the bean
- *foreground:* foregroundcolor of the bean
- *label:* text that displayes the bean
- *font:* font of the text
- *invisible:* hides the bean

4.12.2 Methods:

- *fireReadData():* fires the SerializeDataEvent: readData(mailbeansSerializeData)to all register Listener
- *fireWriteData():* fires the SerializeDataEvent: writeData(mailbeansSerializeData) to all register Listener

OutputListener:

- *handleOutput(mailbeans.OutputListener):* Displays the OutputEvent

4.12.3 Events:

The SettingsDialog bean fires:

- *ProperrtyChangeEvent*

and receives:

- *OutputEvent*