TECHNION Israel Institute of Technology

Faculty of Electrical Engineering

Software Laboratory

Multi-Platform Electronic Organizer with e-mail Client

Performers: Kogan Daniel
Kostas Matvey

Supervisor: Yan Futerman

Winter 2002

Introduction

Project goal

The Project's goal was to design and implement an electronic organizer with built in e-mail client (simplified version of the Microsoft Outlook). The organizer should have been implemented in Java using Swing library components. The Main features required from the application were:

- Usual mail client capabilities Send/Read/Reply and Forward options
- Automatic signature
- Attachments supported
- Electronic address book with "find" operations
- Import/export of the Address book
- Calendar, events and reminder
- Meetings organizer and special meeting email messages

Techniques and Protocols

Following is a brief description of the main techniques and protocols, that we used in our design and implementation.

1) JavaMail

We used a JavaMail extension package provided by Sun. JavaMail is a collection of classes to enable you to send and receive email in as platform independent a way as possible. Notice that in this context "platform" not only means machine and operating system but email protocol.

Introducing the JavaMail API

The JavaMail™ API is an optional package (standard extension) for reading, composing, and sending electronic messages. You use the package to create *Mail User Agent* (MUA) type programs, similar to Eudora, Pine, and Microsoft Outlook. Its main purpose is not for transporting, delivering, and forwarding messages like sendmail or other *Mail Transfer Agent* (MTA) type programs. In other words, users interact with MUA-type programs to read and write emails. MUAs rely on MTAs to handle the actual delivery.

The JavaMail API is designed to provide protocol-independent access for sending and receiving messages by dividing the API into two parts:

- The first part of the API is the focus of this course. Basically, how to send and receive messages independent of the provider/protocol.
- The second part speaks the protocol-specific languages, like SMTP, POP, IMAP, and NNTP. With the JavaMail API, in order to communicate with a server, you need a *provider* for a protocol. The creation of protocol-specific providers is not covered in this course as Sun provides a sufficient set for free.

Reviewing Related Protocols

Before looking into the JavaMail API specifics, step back and take a look at the protocols used with the API. There are basically four that you'll come to know and love:

- SMTP
- POP
- IMAP
- MIME

You will also run across NNTP and some others. Understanding the basics of all the protocols will help you understand how to use the JavaMail API. While the API is designed to be protocol agnostic, you can't overcome the limitations of the underlying protocols. If a capability isn't supported by a chosen protocol, the JavaMail API doesn't magically add the capability on top of it. (As you'll soon see, this usually is a problem when working with POP.)

SMTP

The Simple Mail Transfer Protocol (SMTP) is the mechanism for delivery of email. In the context of the JavaMail API, your JavaMail-based program will communicate with your company or Internet Service Provider's (ISP's) SMTP server. That SMTP server will relay the message on to the SMTP server of the recipient(s) to eventually be acquired by the user(s) through POP or IMAP. This does not require your SMTP server to be an open relay, as authentication is supported, but it is your responsibility to ensure the SMTP server is configured properly. There is nothing in the JavaMail API for tasks like configuring a server to relay messages or to add and remove email accounts.

POP

POP stands for Post Office Protocol. Currently in version 3, also known as POP3, RFC 1939 defines this protocol. POP is the mechanism most people on the Internet use to get their mail. It defines support for a single mailbox for each user. That is all it does, and that is also the source of most confusion. Much of what people are familiar with when using POP, like the ability to see how many new mail messages they have, are not supported by POP at all. These capabilities are built into programs like Eudora or Microsoft Outlook, which remember things like the last mail received and calculate how many are new for you. So, when using the JavaMail API, if you want this type of information, you have to calculate it yourself.

IMAP

IMAP is a more advanced protocol for receiving messages. Defined in RFC 2060, IMAP stands for Internet Message Access Protocol, and is currently in version 4, also known as IMAP4. When using IMAP, your mail server must support the protocol. You can't just change your program to use IMAP instead of POP and expect everything in IMAP to be supported. Assuming your mail server supports IMAP, your JavaMail-based program can take advantage of users having multiple folders on the server and these folders can be shared by multiple users.

Due to the more advanced capabilities, you might think IMAP would be used by everyone. It isn't. It places a much heavier burden on the mail server, requiring the server to receive the new messages, deliver them to users when requested, *and* maintain them in multiple folders for each user. While this does centralize backups, as users' long-term mail folders get larger and larger, everyone suffers when disk space is exhausted. With POP, saved messages get offloaded from the mail server.

MIME

MIME stands for Multipurpose Internet Mail Extensions. It is not a mail transfer protocol. Instead, it defines the content of what is transferred: the format of the messages, attachments, and so on. There are many different documents that take effect here: RFC 822, RFC 2045, RFC 2046, and RFC 2047. As a user of the JavaMail API, you usually don't need to worry about these formats. However, these formats do exist and are used by your programs.

NNTP and Others

Because of the split of the JavaMail API between provider and everything else, you can easily add support for additional protocols. Sun maintains a list of third-party providers that take advantage of protocols that Sun doesn't provide support for, out-of-the-box. There, you'll find support for NNTP (Network News Transport Protocol) [newsgroups], S/MIME (Secure Multipurpose Internet Mail Extensions), and more.

Installing

There are two versions of the JavaMail API commonly used today: 1.2 and 1.1.3. All the examples in this course will work with both. While 1.2 is the latest, 1.1.3 is the version included with the 1.2.1 version of the Java™ 2 Platform, Enterprise Edition™ (J2EE™), so it is still commonly used. The version of the JavaMail API you want to use affects what you download and install. All will work with JDK™ 1.1.6+, Java 2 Platform, Standard Edition™ (J2SE™) version 1.2.x, and J2SE version 1.3.x.

Note: After installing Sun's JavaMail implementation, you can find many example programs in the `demo` directory.

Installing JavaMail 1.2

To use the JavaMail 1.2 API, download the JavaMail 1.2 implementation, unbundle the `javamail-1_2.zip` file, and add the `mail.jar` file to your CLASSPATH. The 1.2 implementation comes with an SMTP, IMAP4, and POP3 provider besides the core classes.

After installing JavaMail 1.2, install the JavaBeans™ Activation Framework.

Installing JavaMail 1.1.3

To use the JavaMail 1.1.3 API, download the JavaMail 1.1.3 implementation, unbundle the `javamail1_1_3.zip` file, and add the `mail.jar` file to your CLASSPATH. The 1.1.3 implementation comes with an SMTP and IMAP4 provider, besides the core classes.

If you want to access a POP server with JavaMail 1.1.3, download and install a POP3 provider. Sun has one available separate from the JavaMail implementation. After downloading and unbundling `pop31_1_1.zip`, add `pop3.jar` to your CLASSPATH, too.

After installing JavaMail 1.1.3, install the JavaBeans Activation Framework.

Installing the JavaBeans Activation Framework

All versions of the JavaMail API require the JavaBeans Activation Framework. The framework adds support for typing arbitrary blocks of data and handling it accordingly. This doesn't sound like much, but it is your basic MIME-type support found in many browsers and mail tools, today. After downloading the framework, unbundle the `jaf1_0_1.zip` file, and add the `activation.jar` file to your CLASSPATH.

For JavaMail 1.2 users, you should now have added `mail.jar` and `activation.jar` to your CLASSPATH.

For JavaMail 1.1.3 users, you should now have added `mail.jar`, `pop3.jar`, and `activation.jar` to your CLASSPATH. If you have no plans of using POP3, you don't need to add `pop3.jar` to your CLASSPATH.

If you don't want to change the CLASSPATH environment variable, copy the JAR files to your `lib/ext` directory under the Java Runtime Environment (JRE) directory. For instance, for the J2SE 1.3 release, the default directory would be `C:\jdk1.3\jre\lib\ext` on a Windows platform.

Using with the Java 2 Enterprise Edition

If you use J2EE, there is nothing special you have to do to use the basic JavaMail API; it comes with the J2EE classes. Just make sure the `j2ee.jar` file is in your CLASSPATH and you're all set.

For J2EE 1.2.1, the POP3 provider comes separately, so download and follow the steps to include the POP3 provider as shown in Installing JavaMail 1.1.3. J2EE 1.3 users get the POP3 provider with J2EE so do not require the separate installation. Neither installation requires you to install the JavaBeans Activation Framework.

Exercise

1. Setting Up Your JavaMail Environment

Reviewing the Core Classes

Before taking a how-to approach at looking at the JavaMail classes in depth, the following walks you through the core classes that make up the API: `Session`, `Message`, `Address`, `Authenticator`, `Transport`, `Store`, and

`Folder`. All these classes are found in the top-level package for the JavaMail API: `javax.mail`, though you'll frequently find yourself using subclasses found in the `javax.mail.internet` package.

Session

The `Session` class defines a basic mail session. It is through this session that everything else works. The `Session` object takes advantage of a `java.util.Properties` object to get information like mail server, username, password, and other information that can be shared across your entire application.

The constructors for the class are private. You can get a single default session that can be shared with the `getDefaultInstance()` method:

```
Properties props = new Properties();
// fill props with any information
Session session = Session.getDefaultInstance(props, null);
```

Or, you can create a unique session with `getInstance()`:

```
Properties props = new Properties();
// fill props with any information
Session session = Session.getInstance(props, null);
```

In both cases here the `null` argument is an `Authenticator` object which is not being used at this time. More on Authenticator shortly.

In most cases, it is sufficient to use the shared session, even if working with mail sessions for multiple user mailboxes. You can add the username and password combination in at a later step in the communication process, keeping everything separate.

Message

Once you have your `Session` object, it is time to move on to creating the message to send. This is done with a type of `Message`. Being an abstract class, you must work with a subclass, in most cases `javax.mail.internet.MimeMessage`. A `MimeMessage` is a email message that understands MIME types and headers, as defined in the different RFCs. Message headers are restricted to US-ASCII characters only, though non-ASCII characters can be encoded in certain header fields.

To create a `Message`, pass along the `Session` object to the `MimeMessage` constructor:

```
MimeMessage message = new MimeMessage(session);
```

Note: There are other constructors, like for creating messages from RFC822-formatted input streams.

Once you have your message, you can set its parts, as `Message` implements the `Part` interface (with `MimeMessage` implementing `MimePart`). The basic mechanism to set the content is the `setContent()` method, with arguments for the content and the mime type:

```
message.setContent("Hello", "text/plain");
```

If, however, you know you are working with a `MimeMessage` and your message is plain text, you can use its `setText()` method which only requires the actual content, defaulting to the MIME type of text/plain:

```
message.setText("Hello");
```

For plain text messages, the latter form is the preferred mechanism to set the content. For sending other kinds of messages, like HTML messages, use the former. More on HTML messages later though.

For setting the subject, use the `setSubject()` method:

```
message.setSubject("First");
```

Address

Once you've created the `Session` and the `Message`, as well as filled the message with content, it is time to address your letter with an `Address`. Like `Message`, `Address` is an abstract class. You use the `javax.mail.internet.InternetAddress` class.

To create an address with just the email address, pass the email address to the constructor:

```
Address address = new InternetAddress("president@whitehouse.gov");
```

If you want a name to appear next to the email address, you can pass that along to the constructor, too:

```
Address address = new InternetAddress("president@whitehouse.gov", "George
Bush");
```

You will need to create address objects for the message's *from* field as well as the *to* field. Unless your mail server prevents you, there is nothing stopping you from sending a message that appears to be from anyone.

Once you've created the addresses, you connect them to a message in one of two ways. For identifying the sender, you use the `setFrom()` and `setReplyTo()` methods.

```
message.setFrom(address)
```

If your message needs to show multiple from addresses, use the `addFrom()` method:

```
Address address[] = ...;
message.addFrom(address);
```

For identifying the message recipients, you use the `addRecipient()` method. This method requires a `Message.RecipientType` besides the address.

```
message.addRecipient(type, address)
```

The three predefined types of address are:

- Message.RecipientType.TO
- Message.RecipientType.CC
- Message.RecipientType.BCC

So, if the message was to go to the vice president, sending a carbon copy to the first lady, the following would be appropriate:

```
Address toAddress = new InternetAddress("vice.president@whitehouse.gov");
Address ccAddress = new InternetAddress("first.lady@whitehouse.gov");
message.addRecipient(Message.RecipientType.TO, toAddress);
message.addRecipient(Message.RecipientType.CC, ccAddress);
```

The JavaMail API provides no mechanism to check for the validity of an email address. While you can program in support to scan for valid characters (as defined by RFC 822) or verify the MX (mail exchange) record yourself, these are all beyond the scope of the JavaMail API.

Authenticator

Like the `java.net` classes, the JavaMail API can take advantage of an `Authenticator` to access protected resources via a username and password. For the JavaMail API, that resource is the mail server. The JavaMail `Authenticator` is found in the `javax.mail` package and is different from the `java.net` class of the same name. The two don't share the same `Authenticator` as the JavaMail API works with Java 1.1, which didn't have the `java.net` variety.

To use the `Authenticator`, you subclass the abstract class and return a `PasswordAuthentication` instance from the `getPasswordAuthentication()` method. You must register the `Authenticator` with the session when created. Then, your `Authenticator` will be notified when authentication is necessary. You could popup a window or read the username and password from a configuration file (though if not encrypted it is not secure), returning them to the caller as a `PasswordAuthentication` object.

```
Properties props = new Properties();
// fill props with any information
Authenticator auth = new MyAuthenticator();
Session session = Session.getDefaultInstance(props, auth);
```

Transport

The final part of sending a message is to use the `Transport` class. This class speaks the protocol-specific language for sending the message (usually SMTP). It's an abstract class and works something like `Session`. You can use the *default* version of the class by just calling the static `send()` method:

```
Transport.send(message);
```

Or, you can get a specific instance from the session for your protocol, pass along the username and password (blank if unnecessary), send the message, and close the connection:

```
message.saveChanges(); // implicit with send()
Transport transport = session.getTransport("smtp");
transport.connect(host, username, password);
transport.sendMessage(message, message.getAllRecipients());
transport.close();
```

This latter way is best when you need to send multiple messages, as it will keep the connection with the mail server active between messages. The basic `send()` mechanism makes a separate connection to the server for each method call.

Note: To watch the mail commands go by to the mail server, set the debug flag with `session.setDebug(true)`.

Store and Folder

Getting messages starts similarly to sending messages, with a `Session`. However, after getting the session, you connect to a [`Store`](), quite possibly with a username and password or `Authenticator`. Like `Transport`, you tell the `Store` what protocol to use:

```
// Store store = session.getStore("imap");
Store store = session.getStore("pop3");
store.connect(host, username, password);
```

After connecting to the `Store`, you can then get a [`Folder`](), which must be opened before you can read messages from it:

```
Folder folder = store.getFolder("INBOX");
folder.open(Folder.READ_ONLY);
Message message[] = folder.getMessages();
```

For POP3, the only folder available is the `INBOX`. If you are using IMAP, you can have other folders available.

Note: Sun's providers are meant to be smart. While `Message message[] = folder.getMessages();` might look like a slow operation reading every message from the server, only when you actually need to get a part of the message is the message content retrieved.

Once you have a `Message` to read, you can get its content with `getContent()` or write its content to a stream with `writeTo()`. The `getContent()` method only gets the message content, while `writeTo()` output includes headers.

```
System.out.println(((MimeMessage)message).getContent());
```

Once you're done reading mail, close the connection to the folder and store.

```
folder.close(aBoolean);
store.close();
```

The boolean passed to the `close()` method of folder states whether or not to update the folder by removing deleted messages.

Moving On

Essentially, understanding how to use these seven classes is all you need for nearly everything with the JavaMail API. Most of the other capabilities of the JavaMail API build off these seven classes to do something a little different or in a particular way, like if the content is an attachment. Certain tasks, like searching, are isolated, and are discussed later.

Using the JavaMail API

You've seen how to work with the core parts of the JavaMail API. In the following sections you'll find a how-to approach for connecting the pieces to do specific tasks.

Sending Messages

Sending an email message involves getting a session, creating and filling a message, and sending it. You can specify your SMTP server by setting the `mail.smtp.host` property for the `Properties` object passed when getting the `Session`:

```
String host = ...;
String from = ...;
String to = ...;

// Get system properties
Properties props = System.getProperties();

// Setup mail server
props.put("mail.smtp.host", host);

// Get session
Session session = Session.getDefaultInstance(props, null);

// Define message
MimeMessage message = new MimeMessage(session);
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO,
  new InternetAddress(to));
message.setSubject("Hello JavaMail");
message.setText("Welcome to JavaMail");

// Send message
Transport.send(message);
```

You should place the code in a try-catch block, as setting up the message and sending it can throw exceptions.

Exercise

2. Sending Your First Message

Fetching Messages

For reading mail, you get a session, get and connect to an appropriate store for your mailbox, open the appropriate folder, and get your message(s). Also, don't forget to close the connection when done.

```
String host = ...;
String username = ...;
String password = ...;

// Create empty properties
Properties props = new Properties();

// Get session
Session session = Session.getDefaultInstance(props, null);

// Get the store
Store store = session.getStore("pop3");
store.connect(host, username, password);

// Get folder
Folder folder = store.getFolder("INBOX");
folder.open(Folder.READ_ONLY);

// Get directory
Message message[] = folder.getMessages();

for (int i=0, n=message.length; i<n; i++) {
   System.out.println(i + ": " + message[i].getFrom()[0]
     + "\t" + message[i].getSubject());
}

// Close connection
folder.close(false);
store.close();
```

What you do with each message is up to you. The above code block just displays who the message is from and the subject. Technically speaking, the list of from addresses could be empty and the `getFrom()[0]` call could throw an exception.

To display the whole message, you can prompt the user after seeing the from and subject fields, and then call the message's `writeTo()` method if they want to see it.

```
BufferedReader reader = new BufferedReader (
  new InputStreamReader(System.in));
```

```
// Get directory
Message message[] = folder.getMessages();
for (int i=0, n=message.length; i<n; i++) {
  System.out.println(i + ": " + message[i].getFrom()[0]
    + "\t" + message[i].getSubject());

  System.out.println("Do you want to read message? " +
    "[YES to read/QUIT to end]");
  String line = reader.readLine();
  if ("YES".equals(line)) {
    message[i].writeTo(System.out);
  } else if ("QUIT".equals(line)) {
    break;
  }
}
```

Exercise

3. Checking for Mail

Deleting Messages and Flags

Deleting messages involves working with the `Flags` associated with the messages. There are different flags for different states, some system-defined and some user-defined. The predefined flags are defined in the inner class `Flags.Flag` and are listed below:

- `Flags.Flag.ANSWERED`
- `Flags.Flag.DELETED`
- `Flags.Flag.DRAFT`
- `Flags.Flag.FLAGGED`
- `Flags.Flag.RECENT`
- `Flags.Flag.SEEN`
- `Flags.Flag.USER`

Just because a flag exists doesn't mean the flag is supported by all mail servers/providers. For instance, besides deleting messages, the POP protocol supports none of them. Checking for *new* mail is not a POP task but one built into mail clients. To find out what flags are supported, ask the folder with `getPermanentFlags()`.

To delete messages, you set the message's `DELETED` flag:

```
message.setFlag(Flags.Flag.DELETED, true);
```

Open up the folder in `READ_WRITE` mode first though:

```
folder.open(Folder.READ_WRITE);
```

Then, when you are done processing all messages, close the folder, passing in a true value to *expunge* the deleted messages.

```
folder.close(true);
```

There is an `expunge()` method of `Folder` that can be used to delete the messages. However, it doesn't work for Sun's POP3 provider. Other providers may or may not implement the capabilities. It will more than likely be implemented for IMAP providers. Because POP only supports single access to the mailbox, you have to close the folder to delete the messages with Sun's provider.

To unset a flag, just pass false to the `setFlag()` method. To see if a flag is set, check with `isSet()`.

Authenticating Yourself

You previously learned that you can use an `Authenticator` to prompt for username and password when needed, instead of passing them in as strings. Here you'll actually see how to more fully use authentication.

Instead of connecting to the `Store` with the host, username, and password, you configure the `Properties` to have the host, and tell the `Session` about your custom `Authenticator` instance, as shown here:

```
// Setup properties
Properties props = System.getProperties();
props.put("mail.pop3.host", host);

// Setup authentication, get session
Authenticator auth = new PopupAuthenticator();
Session session = Session.getDefaultInstance(props, auth);

// Get the store
Store store = session.getStore("pop3");
store.connect();
```

You then subclass `Authenticator` and return a `PasswordAuthentication` object from the `getPasswordAuthentication()` method. The following is one such implementation, with a single field for both. As this isn't a Project Swing tutorial, just enter the two parts in the one field, separated by a comma.

```
import javax.mail.*;
import javax.swing.*;
import java.util.*;

public class PopupAuthenticator extends Authenticator {
  public PasswordAuthentication getPasswordAuthentication() {
    String username, password;

    String result = JOptionPane.showInputDialog(
      "Enter 'username,password'");

    StringTokenizer st = new StringTokenizer(result, ",");
    username = st.nextToken();
    password = st.nextToken();

    return new PasswordAuthentication(username, password);
  }

}
```

Since the `PopupAuthenticator` relies on Swing, it will start up the event-handling thread for AWT. This basically requires you to add a call to `System.exit()` in your code to stop the program.

Replying to Messages

The `Message` class includes a `reply()` method to configure a new `Message` with the proper recipient and subject, adding "Re: " if not already there. This does not add any content to the message, only copying the *from* or *reply-to* header to the new recipient. The method takes a boolean parameter indicating whether to reply to only the sender (false) or reply to all (true).

```
MimeMessage reply = (MimeMessage)message.reply(false);
reply.setFrom(new InternetAddress("president@whitehouse.gov"));
reply.setText("Thanks");
Transport.send(reply);
```

To configure the *reply-to* address when sending a message, use the `setReplyTo()` method.

Exercise

4. Replying to Mail

Forwarding Messages

Forwarding messages is a little more involved. There is no single method to call, and you build up the message to forward by working with the parts that make up a message.

A mail message can be made up of multiple parts. Each part is a `BodyPart`, or more specifically, a `MimeBodyPart` when working with MIME messages. The different body parts get combined into a container called

`Multipart` or, again, more specifically a `MimeMultipart`. To forward a message, you create one part for the text of your message and a second part with the message to forward, and combine the two into a multipart. Then you add the multipart to a properly addressed message and send it.

That's essentially it. To copy the content from one message to another, just copy over its `DataHandler`, a class from the JavaBeans Activation Framework.

```
// Create the message to forward
Message forward = new MimeMessage(session);

// Fill in header
forward.setSubject("Fwd: " + message.getSubject());
forward.setFrom(new InternetAddress(from));
forward.addRecipient(Message.RecipientType.TO,
  new InternetAddress(to));

// Create your new message part
BodyPart messageBodyPart = new MimeBodyPart();
messageBodyPart.setText(
  "Here you go with the original message:\n\n");

// Create a multi-part to combine the parts
Multipart multipart = new MimeMultipart();
multipart.addBodyPart(messageBodyPart);

// Create and fill part for the forwarded content
messageBodyPart = new MimeBodyPart();
messageBodyPart.setDataHandler(message.getDataHandler());

// Add part to multi part
multipart.addBodyPart(messageBodyPart);

// Associate multi-part with message
forward.setContent(multipart);

// Send message
Transport.send(forward);
```

Working with Attachments

Attachments are resources associated with a mail message, usually kept outside of the message like a text file, spreadsheet, or image. As with common mail programs like Eudora and pine, you can *attach* resources to your mail message with the JavaMail API and get those attachments when you receive the message.

Sending Attachments

Sending attachments is quite like forwarding messages. You build up the parts to make the complete message. After the first part, your message text, you add other parts where the `DataHandler` for each is your attachment, instead of the shared handler in the case of a forwarded message. If you are reading the attachment from a file, your attachment data source is a `FileDataSource`. Reading from a URL, it is a `URLDataSource`. Once you have your `DataSource`, just pass it on to the `DataHandler` constructor, before finally attaching it to the `BodyPart` with `setDataHandler()`. Assuming you want to retain the original filename for the attachment, the last thing to do is to set the filename associated with the attachment with the `setFileName()` method of `BodyPart`. All this is shown here:

```
// Define message
Message message = new MimeMessage(session);
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO,
  new InternetAddress(to));
message.setSubject("Hello JavaMail Attachment");

// Create the message part
BodyPart messageBodyPart = new MimeBodyPart();

// Fill the message
messageBodyPart.setText("Pardon Ideas");

Multipart multipart = new MimeMultipart();
multipart.addBodyPart(messageBodyPart);

// Part two is attachment
messageBodyPart = new MimeBodyPart();
DataSource source = new FileDataSource(filename);
messageBodyPart.setDataHandler(new DataHandler(source));
messageBodyPart.setFileName(filename);
multipart.addBodyPart(messageBodyPart);

// Put parts in message
message.setContent(multipart);

// Send the message
Transport.send(message);
```

When including attachments with your messages, if your program is a servlet, your users must upload the attachment besides tell you where to send the message. Uploading each file can be handled with a form encoding type of `multipart/form-data`.

```
<FORM ENCTYPE="multipart/form-data"
    method=post action="/myservlet">
  <INPUT TYPE="file" NAME="thefile">
  <INPUT TYPE="submit" VALUE="Upload">
</FORM>
```

Note: Message size is limited by your SMTP server, not the JavaMail API. If you run into problems, consider increasing the Java heap size by setting the `ms` and `mx` parameters.

Exercise

5. Sending Attachments

Getting Attachments

Getting attachments out of your messages is a little more involved then sending them, as MIME has no simple notion of attachments. The content of your message is a `Multipart` object when it has attachments. You then need to process each `Part`, to get the main content and the attachment(s). Parts marked with a disposition of `Part.ATTACHMENT` from `part.getDisposition()` are clearly attachments. However, attachments can also come across with no disposition (and a non-text MIME type) or a disposition of `Part.INLINE`. When the disposition is either `Part.ATTACHMENT` or `Part.INLINE`, you can save off the content for that message part. Just get the original filename with `getFileName()` and the input stream with `getInputStream()`.

```
Multipart mp = (Multipart)message.getContent();

for (int i=0, n=multipart.getCount(); i<n; i++) {
  Part part = multipart.getBodyPart(i));

  String disposition = part.getDisposition();

  if ((disposition != null) &&
      ((disposition.equals(Part.ATTACHMENT) ||
       (disposition.equals(Part.INLINE))) {
    saveFile(part.getFileName(), part.getInputStream());
  }
}
```

The `saveFile()` method just creates a `File` from the filename, reads the bytes from the input stream, and writes them off to the file. In case the file already exists, a number is added to the end of the filename until one is found that doesn't exist.

```
// from saveFile()
File file = new File(filename);
for (int i=0; file.exists(); i++) {
  file = new File(filename+i);
}
```

The code above covers the simplest case where message parts are flagged appropriately. To cover all cases, handle when the disposition is null and get the MIME type of the part to handle accordingly.

```
if (disposition == null) {
  // Check if plain
  MimeBodyPart mbp = (MimeBodyPart)part;
  if (mbp.isMimeType("text/plain")) {
    // Handle plain
  } else {
    // Special non-attachment cases here of image/gif, text/html, ...
  }
...
}
```

Processing HTML Messages

Sending HTML-based messages can be a little more work than sending plain text messages, though it doesn't have to be that much more work. It all depends on your specific requirements.

Sending HTML Messages

If all you need to do is send the equivalent of an HTML file as the message and let the mail reader worry about fetching any embedded images or related pieces, use the `setContent()` method of `Message`, passing along the content as a `String` and setting the content type to `text/html`.

```
String htmlText = "<H1>Hello</H1>" +
  "<img src=\"http://www.jguru.com/images/logo.gif\">";
message.setContent(htmlText, "text/html"));
```

On the receiving end, if you fetch the message with the JavaMail API, there is nothing built into the API to display the message as HTML. The JavaMail API only sees it as a stream of bytes. To display the message as HTML, you must either use the Swing `JEditorPane` or some third-party HTML viewer component.

```
if (message.getContentType().equals("text/html")) {
  String content = (String)message.getContent();
  JFrame frame = new JFrame();
  JEditorPane text = new JEditorPane("text/html", content);
  text.setEditable(false);
  JScrollPane pane = new JScrollPane(text);
  frame.getContentPane().add(pane);
  frame.setSize(300, 300);
  frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
  frame.show();
}
```

Including Images with Your Messages

On the other hand, if you want your HTML content message to be complete, with embedded images included as part of the message, you must treat the image as an attachment and reference the image with a special `cid` URL, where the `cid` is a reference to the `Content-ID` header of the image attachment.

The process of embedding an image is quite similar to attaching a file to a message, the only difference is that you have to tell the `MimeMultipart` that the parts are related by setting its subtype in the constructor (or with `setSubType()`) and set the `Content-ID` header for the image to a random string which is used as the `src` for the image in the `img` tag. The following demonstrates this completely.

```
String file = ...;

// Create the message
Message message = new MimeMessage(session);

// Fill its headers
message.setSubject("Embedded Image");
message.setFrom(new InternetAddress(from));
message.addRecipient(Message.RecipientType.TO,
  new InternetAddress(to));

// Create your new message part
BodyPart messageBodyPart = new MimeBodyPart();
String htmlText = "<H1>Hello</H1>" +
  "<img src=\"cid:memememe\">";
messageBodyPart.setContent(htmlText, "text/html");

// Create a related multi-part to combine the parts
MimeMultipart multipart = new MimeMultipart("related");
multipart.addBodyPart(messageBodyPart);

// Create part for the image
messageBodyPart = new MimeBodyPart();

// Fetch the image and associate to part
DataSource fds = new FileDataSource(file);
messageBodyPart.setDataHandler(new DataHandler(fds));
messageBodyPart.setHeader("Content-ID","memememe");

// Add part to multi-part
multipart.addBodyPart(messageBodyPart);

// Associate multi-part with message
message.setContent(multipart);
```

Exercise

6. Sending HTML Messages with Images

Searching with SearchTerm

The JavaMail API includes a filtering mechanism found in the `javax.mail.search` package to build up a `SearchTerm`. Once built, you then ask a `Folder` what messages match, retrieving an array of `Message` objects:

```
SearchTerm st = ...;
Message[] msgs = folder.search(st);
```

There are 22 different classes available to help you build a search term.

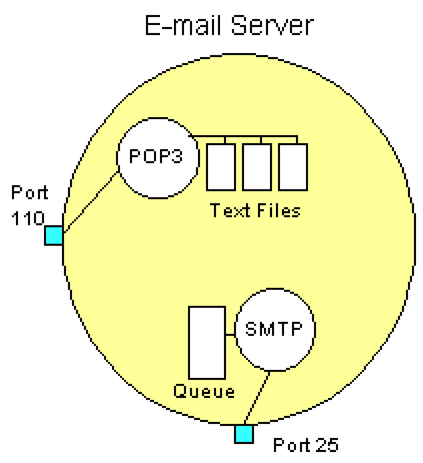
- AND terms (class `AndTerm`)
- OR terms (class `OrTerm`)
- NOT terms (class `NotTerm`)
- SENT DATE terms (class `SentDateTerm`)
- CONTENT terms (class `BodyTerm`)
- HEADER terms (`FromTerm` / `FromStringTerm`, `RecipientTerm` / `RecipientStringTerm`, `SubjectTerm`, etc.)

Essentially, you build up a logical expression for matching messages, then search. For instance the following term searches for messages with a (partial) subject string of `ADV` or a from field of `friend@public.com`. You might consider periodically running this query and automatically deleting any messages returned.

```
SearchTerm st =
  new OrTerm(
    new SubjectTerm("ADV:"),
    new FromStringTerm("friend@public.com"));
Message[] msgs = folder.search(st);
```

The Real E-mail System

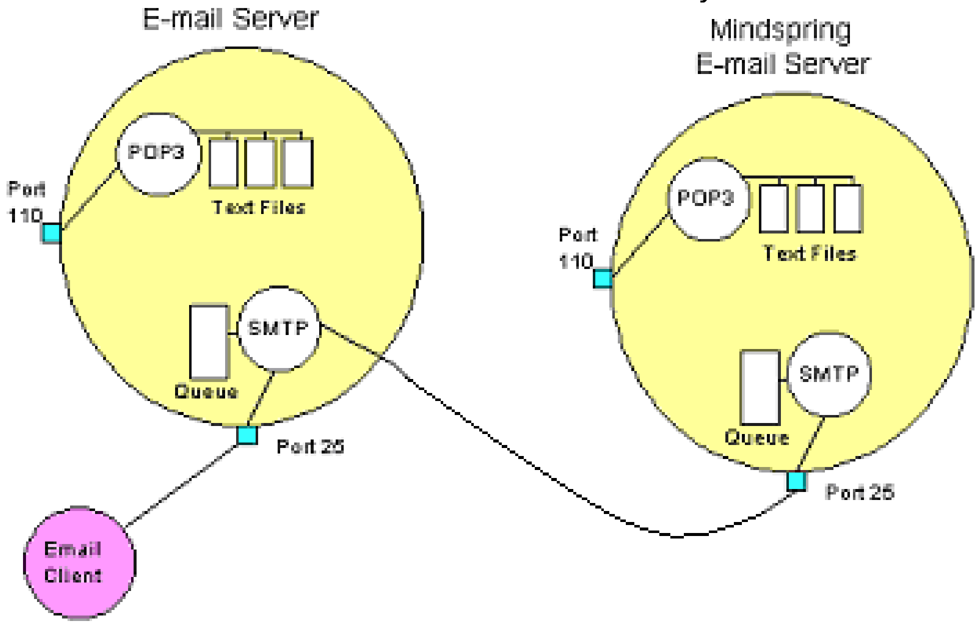
For the vast majority of people right now, the real e-mail system consists of two different servers running on a server machine. One is called the **SMTP Server**, where SMTP stands for Simple Mail Transfer Protocol. The SMTP server handles outgoing mail. The other is a **POP3 Server**, where POP stands for Post Office Protocol. The POP3 server handles incoming mail. A typical e-mail server looks like this:



The SMTP server listens on well-known port number 25, while POP3 listens on port 110.

The SMTP Server

Whenever you send a piece of e-mail, your e-mail client interacts with the SMTP server to handle the sending. The SMTP server on your host may have conversations with other SMTP servers to actually deliver the e-mail.



Let's assume that you want to send a piece of e-mail. Your e-mail ID is **userA**, and You have your account on **yahoo.com**. You want to send e-mail to **userB@startup.com**. You are using a stand-alone e-mail client like our MultEMail or Outlook Express.

When You set up your account at yahoo, You told MultEMail the name of the mail server -- **mail.yahoo.com**. When You compose a message and press the Send button, here is what happens:

1. MultEMail connects to the SMTP server at mail.yahoo.com using **port 25**.
2. MultEMail has a conversation with the SMTP server, telling the SMTP server the address of the sender and the address of the recipient, as well as the body of the message.
3. The SMTP server takes the "to" address (userB@startup.com) and breaks it into two parts:
    - The recipient name (userB)
    - The domain name (startup.com)

   If the "to" address had been another user at yahoo.com, the SMTP server would simply hand the message to the POP3 server for yahoo.com (using a little program called the **delivery agent**). Since the recipient is at another domain, SMTP needs to communicate with that domain.

4. The SMTP server has a conversation with a **Domain Name Server**, or **DNS**. It says, "Can you give me the IP address of the SMTP server for startup.com?" The DNS replies with the one or more IP addresses for the SMTP server(s) that Startup server operates.
5. The SMTP server at yahoo.com connects with the SMTP server at Startup using port 25. It has the same simple text conversation that your e-mail client had with the SMTP server for Yahoo, and gives the message to the Startup server. The Startup server recognizes that the domain name for userB is at Startup, so it hands the message to Startup's POP3 server, which puts the message in userB's mailbox.

If, for some reason, the SMTP server at Yahoo cannot connect with the SMTP server at Startup, then the message goes into a queue. The SMTP server on most machines uses a program called **sendmail** to do the actual sending, so this queue is called the **sendmail queue**. Sendmail will periodically try to resend the messages in its queue. For example, it might retry every 15 minutes. After four hours, it will usually send you a piece of mail that tells you there is some sort of problem. After five days, most sendmail configurations give up and return the mail to you undelivered.

The actual conversation that an e-mail client has with an SMTP server is incredibly simple and human readable. It is specified in public documents called **Requests For Comments** (RFC), and a typical conversation looks something like this:

```
helo test
250 mx1.startup.com Hello abc.sample.com
[220.57.69.37], pleased to meet you
mail from: test@sample.com
250 2.1.0 test@sample.com... Sender ok
```

```
rcpt to: userB@startup.com
250 2.1.5 userB... Recipient ok
data
354 Enter mail, end with "." on a line by itself
from: test@sample.com
to: userB@startup.com
subject: testing
John, I am testing...
.
250 2.0.0 e1NMajH24604 Message accepted
for delivery
quit
221 2.0.0 mx1.startup.com closing connection
Connection closed by foreign host.
```

What the e-mail client says is in red, and what the SMTP server replies is in green. The e-mail client introduces itself, indicates the "from" and "to" addresses, delivers the body of the message and then quits. You can, in fact, **telnet** to a mail server machine at port 25 and have one of these dialogs yourself -- this is how people "spoof" e-mail.

You can see that the SMTP server understands very simple text commands like HELO, MAIL, RCPT and DATA. The most common commands are:

- **HELO** - introduce yourself
- **EHLO** - introduce yourself and request extended mode
- **MAIL FROM:** - specify the sender
- **RCPT TO:** - specify the recipient
- **DATA** - specify the body of the message (To:, From: and Subject: should be the first three lines.)
- **RSET** - reset
- **QUIT** - quit the session
- **HELP** - get help on commands
- **VRFY** - verify an address
- **EXPN** - expand an address
- **VERB** - verbose

The POP3 Server

In the simplest implementations of POP3, the server really does maintain a collection of text files -- one for each e-mail account. When a message arrives, the POP3 server simply appends it to the bottom of the recipient's file!

When you check your e-mail, your e-mail client connects to the POP3 server using **port 110**. The POP3 server requires an **account name** and a **password**. Once you have logged in, the POP3 server opens your text file and allows you to access it. Like the SMTP server, the POP3 server understands a very simple set of text commands. Here are the most common commands:

- **USER** - enter your user ID
- **PASS** - enter your password
- **QUIT** - quit the POP3 server
- **LIST** - list the messages and their size
- **RETR** - retrieve a message, pass it a message number
- **DELE** - delete a message, pass it a message number
- **TOP** - show the top x lines of a message, pass it a message number and the number of lines

Your e-mail client connects to the POP3 server and issues a series of commands to bring copies of your e-mail messages to your local machine. Generally, it will then delete the messages from the server (unless you've told the e-mail client not to).

You can see that the POP3 server simply acts as an interface between the e-mail client and the text file containing your messages. And again, you can see that the POP3 server is extremely simple! You can connect to it through telnet at port 110 and issue the commands yourself if you would like to.

2) Swing

Sun Microsystems is leveraging the technology of Netscape™ Communications, IBM, and Lighthouse Design (now owned by Sun) to create a set of Graphical User Interface (GUI) classes that integrate with JDK™ 1.1.5+, are standard with the Java ® 2 platform and provide a more polished look and feel than the standard AWT component set. The collection of APIs coming out of this effort, called the Java Foundation Classes (JFC), allows developers to build full-featured enterprise-ready applications.

JFC is composed of five APIs: AWT, Java™ 2D, Accessibility, Drag and Drop, and Swing. The AWT components refer to the AWT as it exists in JDK versions 1.1.2 and later. Java 2D is a graphics API based on technology licensed from IBM/Taligent. It is currently available with the Java® 2 Platform (and not usable with JDK 1.1). The Accessibility API provides assistive technologies, like screen magnifiers, for use with the various pieces of JFC. Drag and Drop support is part of the next JavaBean™ generation, "Glasgow," and is also available with the Java® 2 platform.

Swing includes a component set that is targeted at forms-based applications. Loosely based on Netscape's acclaimed Internet Foundation Classes (IFC), the Swing components have had the most immediate impact on Java development. They provide a set of well-groomed widgets and a framework to specify how GUIs are visually presented, independent of platform. At the time this was written, the Swing release is at 1.1 (FCS).

IFC, AWT, and Swing: Sorting it all out

Though the Swing widgets were based heavily on IFC, the two APIs bear little resemblance to one another from the perspective of a developer. The look and feel of some Swing widgets and their rendering is primarily what descended from IFC, although you may notice some other commonalties.

The AWT 1.1 widgets and event model are still present for the Swing widgets. However, the 1.0 event model does not work with Swing widgets. The Swing widgets simply extend AWT by adding a new set of components, the *JComponents*, and a group of related support classes. As with AWT, Swing components are all JavaBeans and participate in the JavaBeans event model.

A subset of Swing widgets is analogous to the basic AWT widgets. In some cases, the Swing versions are simply lightweight components, rather than peer-based components. The lightweight component architecture was introduced in AWT 1.1. It allows components to exist without native operating system widgets. Instead, they participate in the Model/View/Controller (MVC) architecture, which will be described in Part II of this course. Swing also contains some new widgets such as trees, tabbed panes, and splitter panes that will greatly improve the look and functionality of GUIs.

Swing Package Overview

Swing can expand and simplify your development of cross-platform applications. The Swing collection consists of seventeen packages, each of which has its own distinct purpose. As you'll learn in this short course, these packages make it relatively easy for you to put together a variety of applications that have a high degree of sophistication and user friendliness.

`javax.swing`
: The high level swing package primarily consists of components, adapters, default component models, and interfaces for all the delegates and models.

`javax.swing.border`
: The border package declares the `Border` interface and classes, which define specific border rendering styles.

`javax.swing.colorchooser`
: The colorchooser package contains support classes for the color chooser component.

`javax.swing.event`

The event package is for the Swing-specific event types and listeners. In addition to the `java.awt.event` types, Swing components can generate their own event types.

`javax.swing.filechooser`

The filechooser package contains support classes for the file chooser component.

`javax.swing.plaf.*`

The pluggable look-and-feel (PLAF) packages contain the User Interface (UI) classes (delegates) which implement the different look-and-feel aspects for Swing components. There are also PLAF packages under the `javax.swing.plaf` hierarchy.

`javax.swing.table`

The table package contains the support interfaces and classes the Swing table component.

`javax.swing.text`

The text package contains the support classes for the Swing document framework.

`javax.swing.text.html.*`

The text.html package contains the support classes for an HTML version 3.2 renderer and parser.

`javax.swing.text.rtf`

The text.rtf package contains the support classes for a basic Rich Text Format (RTF) renderer.

`javax.swing.tree`

The tree package contains the interfaces and classes which support the Swing tree component.

`javax.swing.undo`

The undo package provides the support classes for implementing undo/redo capabilities in a GUI.

`javax.accessibility`

The JFC Accessibility package is included with the Swing classes. However, its usage is not discussed here.

Design and Implementation

In this chapter we’ll examine the structure and the behavior of our application, specifying all the features and points of interest from the programmer’s point of view. The chapter “User guide” deals with a pure operational description of an application, at the level that matches an inexperienced user.

Getting Messages

We use only POP3 protocol for receiving of messages. This means that if a user configures our client to get mail from server that does not support POP3 protocol (and, for instance, supports the more sophisticated protocol – the IMAP) – the application will fail to connect to that server and will show the user an error message. Generally, for successful receiving, the user must provide the application with the following information:

- The name or the IP-address of the POP3 server on which user's mailbox is located
- His username on that server
- The password for that username

Additional receiving options (not obligatory) are:

- "Save password" option. If the user is convinced that no unauthorized person has access to his computer, he may use the "save password" option, and his password will be saved in the system until he unchecks that option.
- "Don't delete mail from the server" option. This is a very important option. When checked, the application will leave all the messages on the server after it gets them to the local workstation. It is useful in the case that the user is running the application on station A, and he wants to be able to get the same messages later on a machine B. However, one must remember, that most of the mail servers set a specific quota on a mailbox size, and if the mail is not periodically removed from the mailbox, it will eventually reach its limit… When the option is unchecked, all the messages are removed from the server once they are downloaded to the client's computer.

The type of e-mail messages our application deals with is the most standard kind of Mime messages. Mime, of course, also supports very complex, hierarchically structured messages, whose parts may be not only "plain text" parts, but also HTML texts, and many other types. Again, Mult-E-Mail supports only the messages with the following content: one plain text body, and unlimited number of file attachments. From our observation, more than 95% of e-mail messages on the net fall into that category. In case that the message is not a standard one, the application will show it anyway, but probably represented in an inaccurate way (for example, HTML text will appear as a plain text, with all the tags ignored, and shown as they are).

When the user clicks on “Get Mail” button, the connection application (with the use of package) connects to the specified POP3 server, provides it with the user’s details (username and password) and fetches <u>all</u> the messages from the INBOX folder in the user’s Mailbox. Between all the fetched messages the program looks only for those that are not already in its database. Those messages are processed, saved into the database and shown to the user as new rows in the list.

There are three different “lists” (or “folders”) that contain messages:

- ***In*** folder - contains all the mail messages that were received from the server, and weren’t deleted or moved to Temp folder.
- ***Out*** folder – holds all the messages that were sent by the user, and weren’t deleted afterwards.

- ***Temp*** folder – holds the messages that were “moved” into it from the In folder and weren’t deleted since.

Messages that have been read (opened for viewing) by the user at least once, may be distinguished from the unread messages: in all folders, rows that represent them are colored in gray, instead of black.

Address Book

One of the most basic features of our application is the Address Book. This is, actually, a database of address entries with the common operations available on it:

- Modifying the DB: a user can add a new address entry, remove an existing one, or change some of the data of a specific entry.
- Searching the DB. Three kinds of search are available:

    - Quick search: this search works as following: in order to find some person in the address book (for example, “Tamara Armishev”), the user should start typing his name, and the application will select the entry whose name starts with the typed sequence. The problem of this search is that the user must remember what person's name is, and how exactly he entered it into the database (if he actually entered it as "Armishev Tamara"), the quick search won't be helpful in our example). That's why we offer a more sophisticated type of search:

- The advanced search: this option allows the user to find entries by many criteria: name, address, phone, and e-mail address. The user may specify some of these fields, and our program will show to him all the entries that match the specified criteria (with the logical AND comparison). Moreover, he doesn't have to specify the full string value of the criteria, but only some part of it. Example:
  The search for the criteria *Name:* "Dan*"*, *Address:* "Haifa"
  will return, for instance, entries:
  Name: "Daniel Kogan", *Address*: "Neve Shanan, Haifa, Israel"
  and *Name*: "Yuri Chemo<u>dan</u>ov", *Address*: "Haifa, IL",
  but <u>won't</u> return *Name*: "Dany", *Address*: "Jerusalem".

- Of course, the user can simply go over all the entries in the table "manually" (by Scrolling) and look for the desired entry.

The Address book can be opened simply by pressing "Address Book" button in the main application window, but it may also be invoked during a mail composition, and used for adding addresses into the *To:/Cc:/Bcc:* fields.

Scheduler

An essential part of Mult-E-Mail is the Scheduler. The term "Scheduler" includes a Calendar, an event manager and a reminder.

An event is something like a memo for the user. It is set for a specific date and time and has a subject and a full description. For example: date "13/08/02 08:00", subject "My wife's birthday", description "Must buy her a nice present".

In order to add a new event, the user must first pick a date on our program's calendar. Beside the calendar he will see all the scheduled events for the chosen date. Then he should open a new entry, pick a time for it, and fill it with a subject and a description. Scheduling several events for the same time is allowed under condition that they have different subjects.

The application, of course, has a mechanism that reminds the user of events. First, the user should open the preferences window and select a time interval "T" (0 or more minutes), which will be used by the scheduler. Once a minute, a special thread is run in the application. That thread checks for events that are going to happen in the next T or less minutes and weren't reminded yet. These events are announced to the user (a sound is played and a window with the event information pops up), and then the event is marked (internally in the program) as an "announced event", so that it will not be announced again. Notice, that the mentioned criterion "T or less minutes" includes also negative values, which means that the event was scheduled in the past (expired). If that event has not yet been announced (the application wasn't running at that time), it is announced to the user now, as an expired event.

Meetings

A special type of messages exists in our application – meetings. Sending a meeting request is, actually, sending a regular mail, but in a specific format. There is a small technical difference in treating meetings: a meeting request window differs from the regular mail composition window - it does not allow the user to add attachments, and it has special components for choosing date and time for the meeting.

A much more important difference is the link that exists between the meetings and the event's scheduler:

- When the user is sending a meeting request, a new event is automatically added to the database – an event that reminds the user himself about that meeting. Obviously, it is scheduled to the meeting's date and time, and has the same subject.

- When the recipients get the meeting request mail (and if they are using Mult-E-Mail program) – their program will recognize this mail as a meeting, and will add a new event that will remind of that meeting.

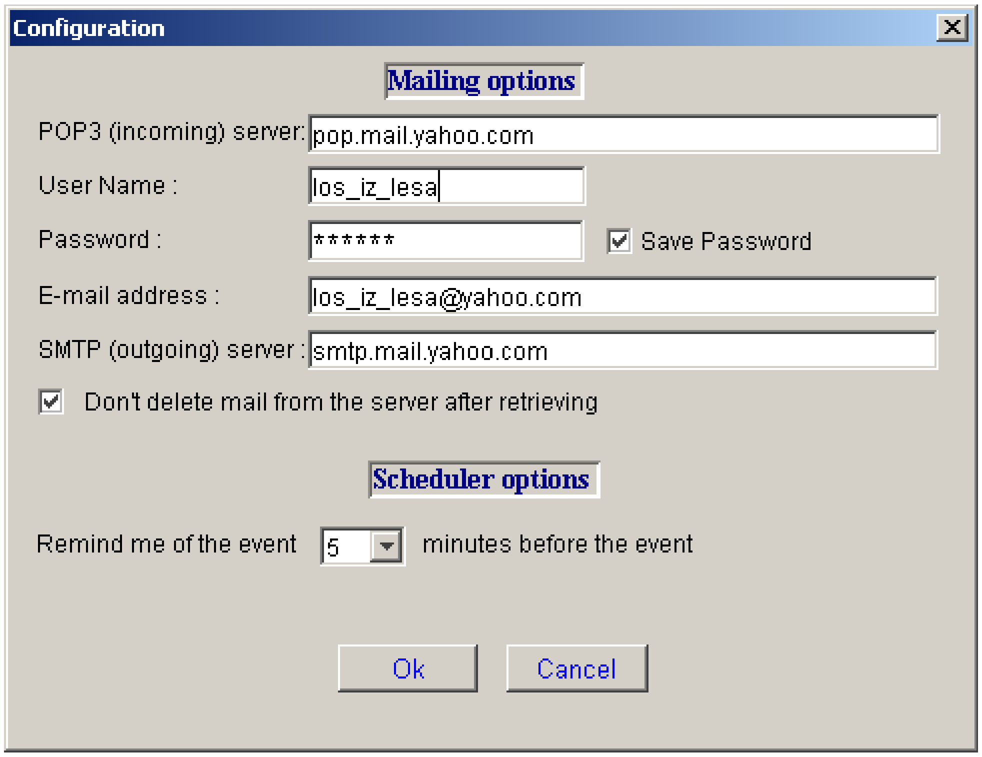
Notice – meetings are ordinary e-mail messages, plus – they sometimes invoke creation of new events in the system.

User Guide

1. Application's windows

First, let us introduce you to all main windows of our application's interface. Each window will get its name, so that in future chapters we'll be able to refer to them easily.

1.1 *Configuration window* – in this window the user will enter the necessary data for the Mailing and the Scheduler configurations :



Configuration

Mailing options

POP3 (incoming) server: pop.mail.yahoo.com

User Name : los_iz_lesa

Password : ****** [✓] Save Password

E-mail address : los_iz_lesa@yahoo.com

SMTP (outgoing) server : smtp.mail.yahoo.com

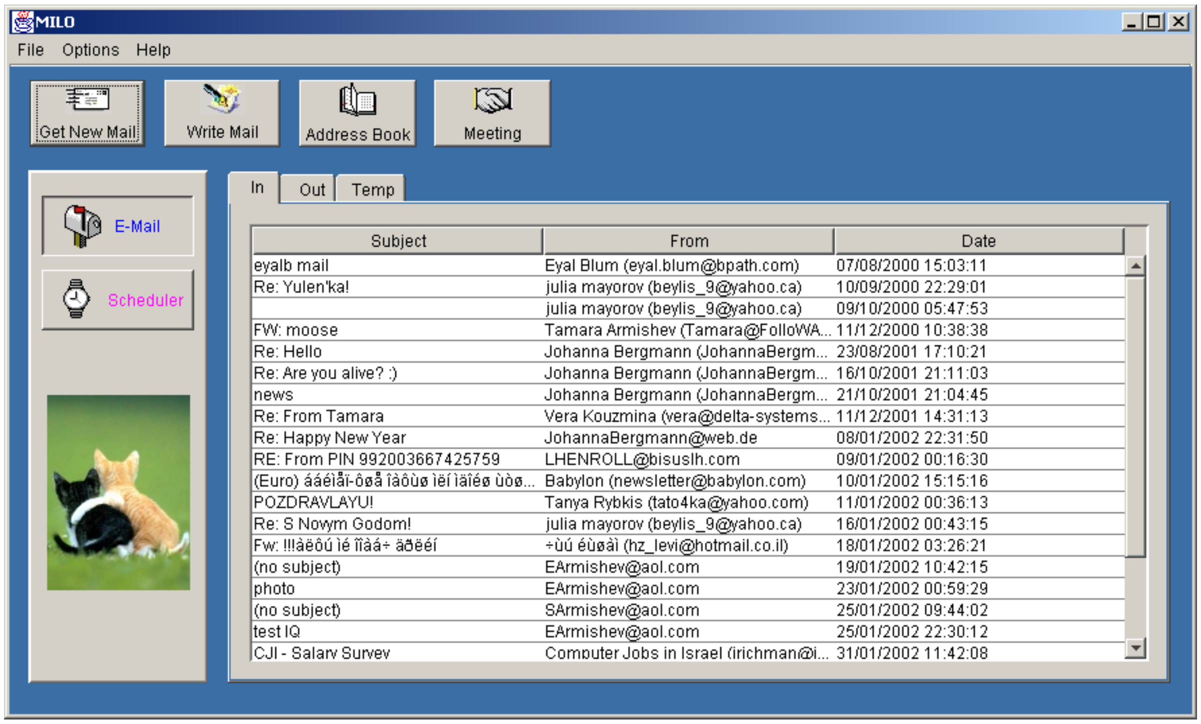
[✓] Don't delete mail from the server after retrieving

Scheduler options

Remind me of the event 5 minutes before the event

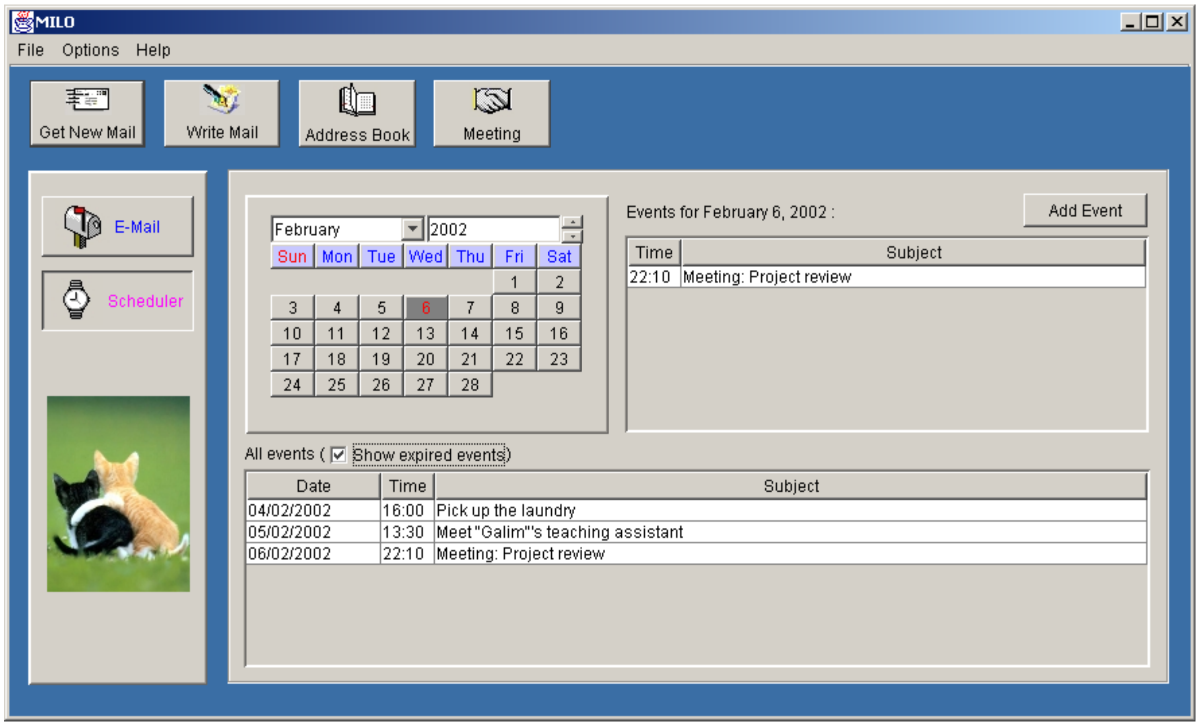
1.2 *Main window* - this window is opened when the application is started. It may have two states - When the button "E-Mail" is pressed, then we'll call

the window "Main window in E-Mail state":



Subject	From	Date
eyalb mail	Eyal Blum (eyal.blum@bpath.com)	07/08/2000 15:03:11
Re: Yulen'ka!	julia mayorov (beylis_9@yahoo.ca)	10/09/2000 22:29:01
	julia mayorov (beylis_9@yahoo.ca)	09/10/2000 05:47:53
FW: moose	Tamara Armishev (Tamara@FolloWA...	11/12/2000 10:38:38
Re: Hello	Johanna Bergmann (JohannaBergm...	23/08/2001 17:10:21
Re: Are you alive? :)	Johanna Bergmann (JohannaBergm...	16/10/2001 21:11:03
news	Johanna Bergmann (JohannaBergm...	21/10/2001 21:04:45
Re: From Tamara	Vera Kouzmina (vera@delta-systems...	11/12/2001 14:31:13
Re: Happy New Year	JohannaBergmann@web.de	08/01/2002 22:31:50
RE: From PIN 992003667425759	LHENROLL@bisuslh.com	09/01/2002 00:16:30
(Euro) ááéìåï-ôøå ìàôùø ìëí ìàîéø ùòø...	Babylon (newsletter@babylon.com)	10/01/2002 15:15:16
POZDRAVLAYU!	Tanya Rybkis (tato4ka@yahoo.com)	11/01/2002 00:36:13
Re: S Nowym Godom!	julia mayorov (beylis_9@yahoo.ca)	16/01/2002 00:43:15
Fw: !!!àëôú ìé ïîàá÷ äðëéí	÷ùú éùøàì (hz_levi@hotmail.co.il)	18/01/2002 03:26:21
(no subject)	EArmishev@aol.com	19/01/2002 10:42:15
photo	EArmishev@aol.com	23/01/2002 00:59:29
(no subject)	SArmishev@aol.com	25/01/2002 09:44:02
test IQ	EArmishev@aol.com	25/01/2002 22:30:12
CJI - Salary Survey	Computer Jobs in Israel (irichman@i...	31/01/2002 11:42:08

When the button "Scheduler" is pressed – we'll call the window "Main window in a Scheduler state":



Sun	Mon	Tue	Wed	Thu	Fri	Sat
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28		

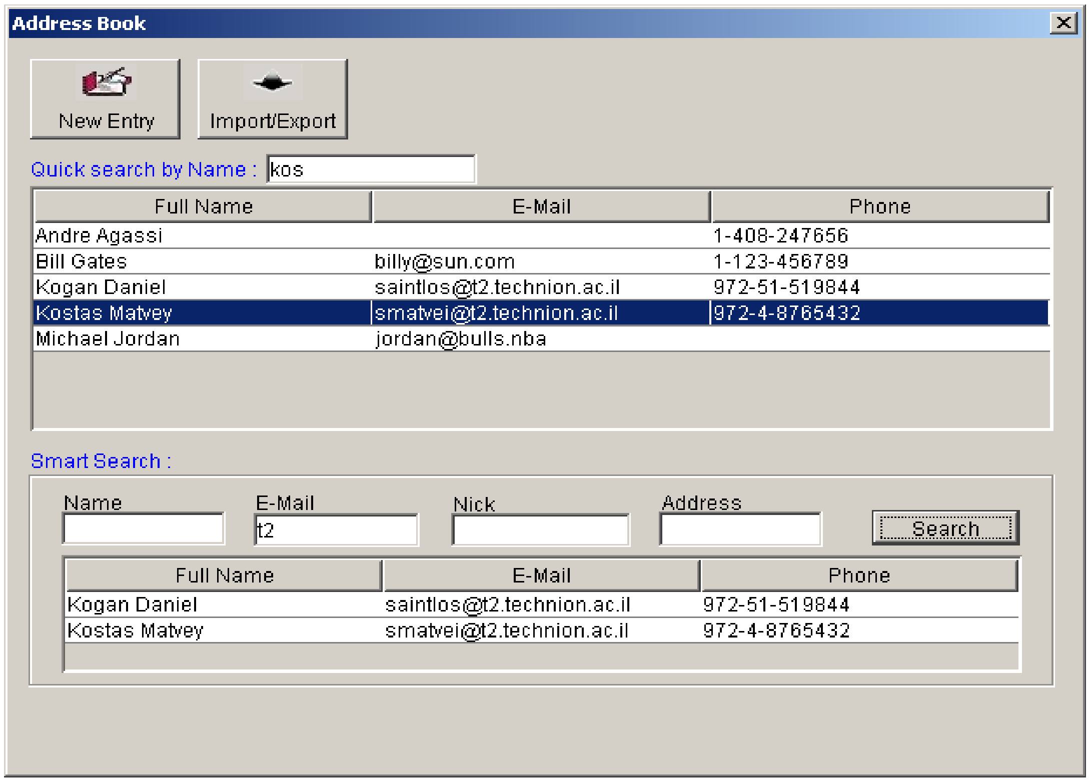
Events for February 6, 2002 :

Time	Subject
22:10	Meeting: Project review

All events (☑ Show expired events)

Date	Time	Subject
04/02/2002	16:00	Pick up the laundry
05/02/2002	13:30	Meet "Galim"'s teaching assistant
06/02/2002	22:10	Meeting: Project review

1.3 *Address Book window –* this is the window that provides an interface to the Address Book database and operations on it:

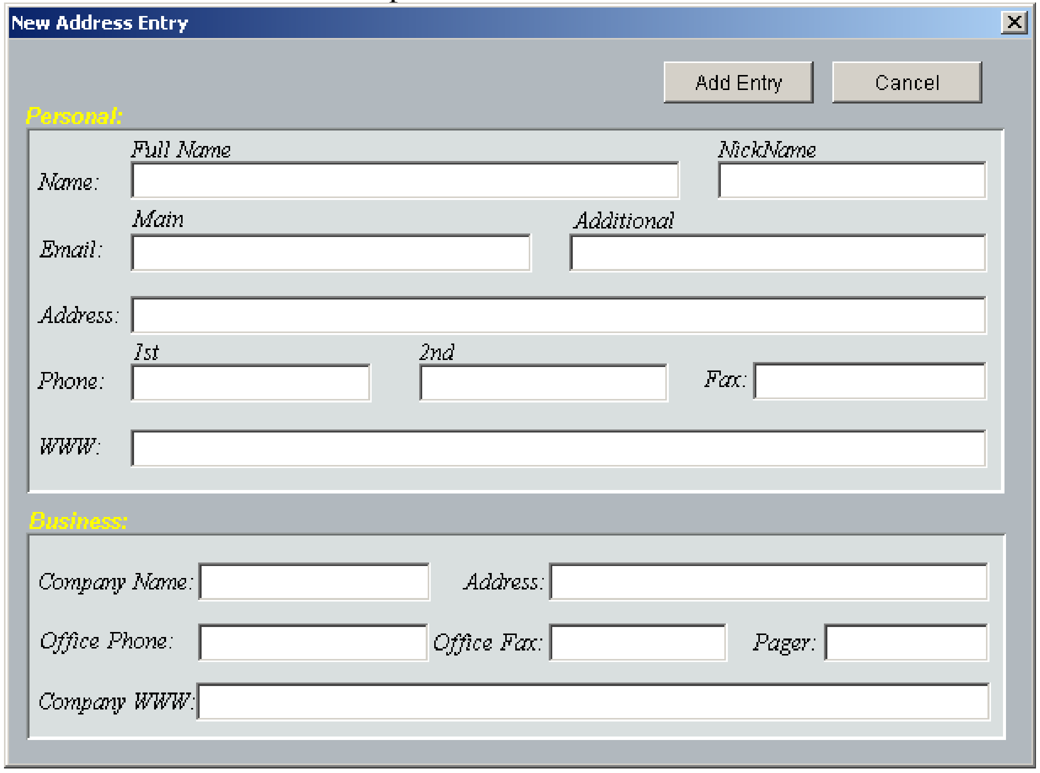


Full Name	E-Mail	Phone
Andre Agassi		1-408-247656
Bill Gates	billy@sun.com	1-123-456789
Kogan Daniel	saintlos@t2.technion.ac.il	972-51-519844
Kostas Matvey	smatvei@t2.technion.ac.il	972-4-8765432
Michael Jordan	jordan@bulls.nba	

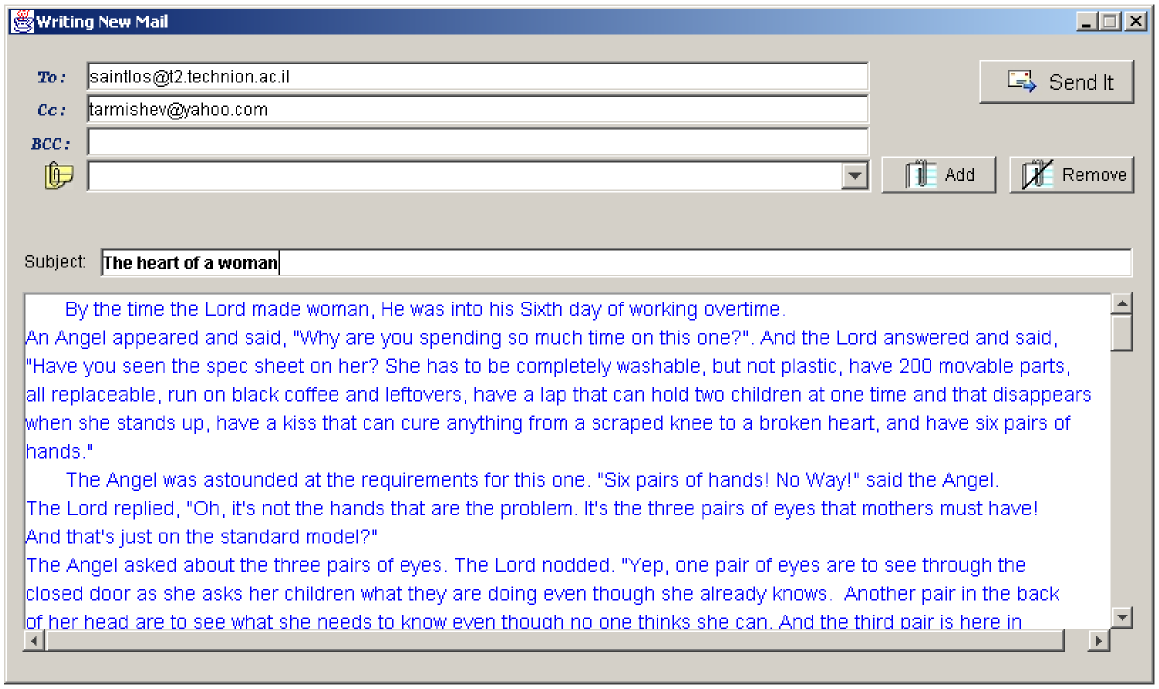
Name	E-Mail	Nick	Address
	t2		

Full Name	E-Mail	Phone
Kogan Daniel	saintlos@t2.technion.ac.il	972-51-519844
Kostas Matvey	smatvei@t2.technion.ac.il	972-4-8765432

1.4 *New Address Entry window –* this window collects from the user information about the new person added to the DB :



1.5 *Mail Composition window –* this is a window used for composing and sending an e-mail message:



To : saintlos@t2.technion.ac.il

Cc : tarmishev@yahoo.com

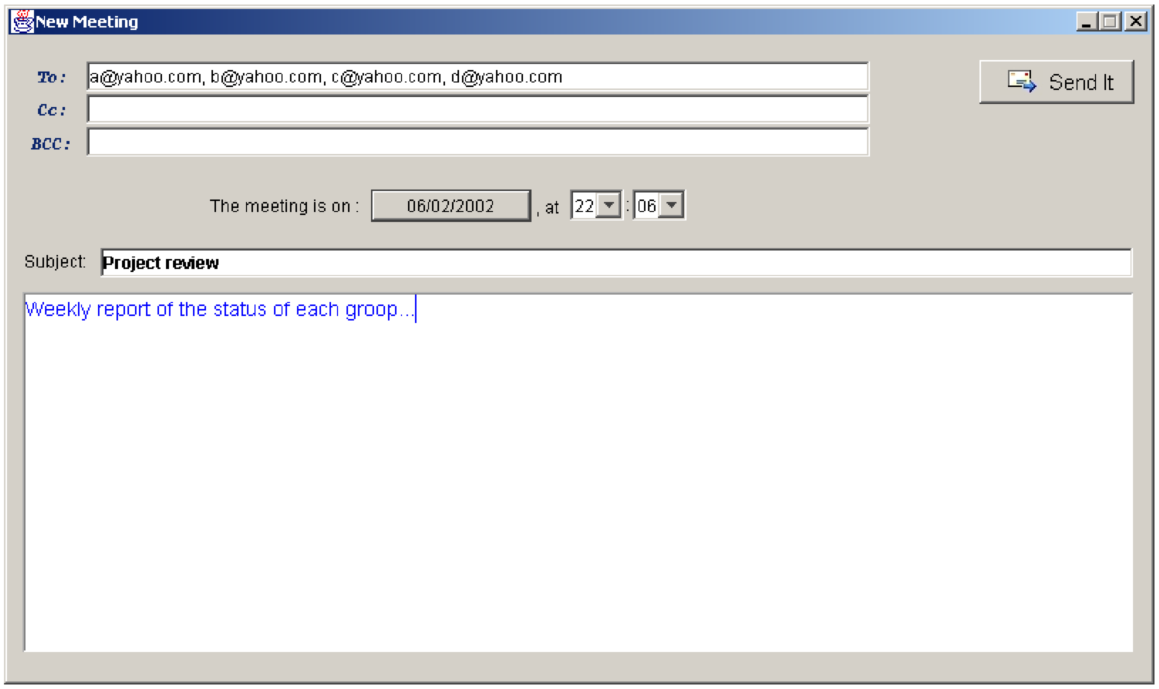
BCC :

Subject: The heart of a woman

By the time the Lord made woman, He was into his Sixth day of working overtime.
An Angel appeared and said, "Why are you spending so much time on this one?". And the Lord answered and said, "Have you seen the spec sheet on her? She has to be completely washable, but not plastic, have 200 movable parts, all replaceable, run on black coffee and leftovers, have a lap that can hold two children at one time and that disappears when she stands up, have a kiss that can cure anything from a scraped knee to a broken heart, and have six pairs of hands."

The Angel was astounded at the requirements for this one. "Six pairs of hands! No Way!" said the Angel.
The Lord replied, "Oh, it's not the hands that are the problem. It's the three pairs of eyes that mothers must have! And that's just on the standard model?"
The Angel asked about the three pairs of eyes. The Lord nodded. "Yep, one pair of eyes are to see through the closed door as she asks her children what they are doing even though she already knows. Another pair in the back of her head are to see what she needs to know even though no one thinks she can. And the third pair is here in

1.6 *Meeting Composition window* – this is a window used for composing and sending a meeting message:



To : a@yahoo.com, b@yahoo.com, c@yahoo.com, d@yahoo.com

Cc :

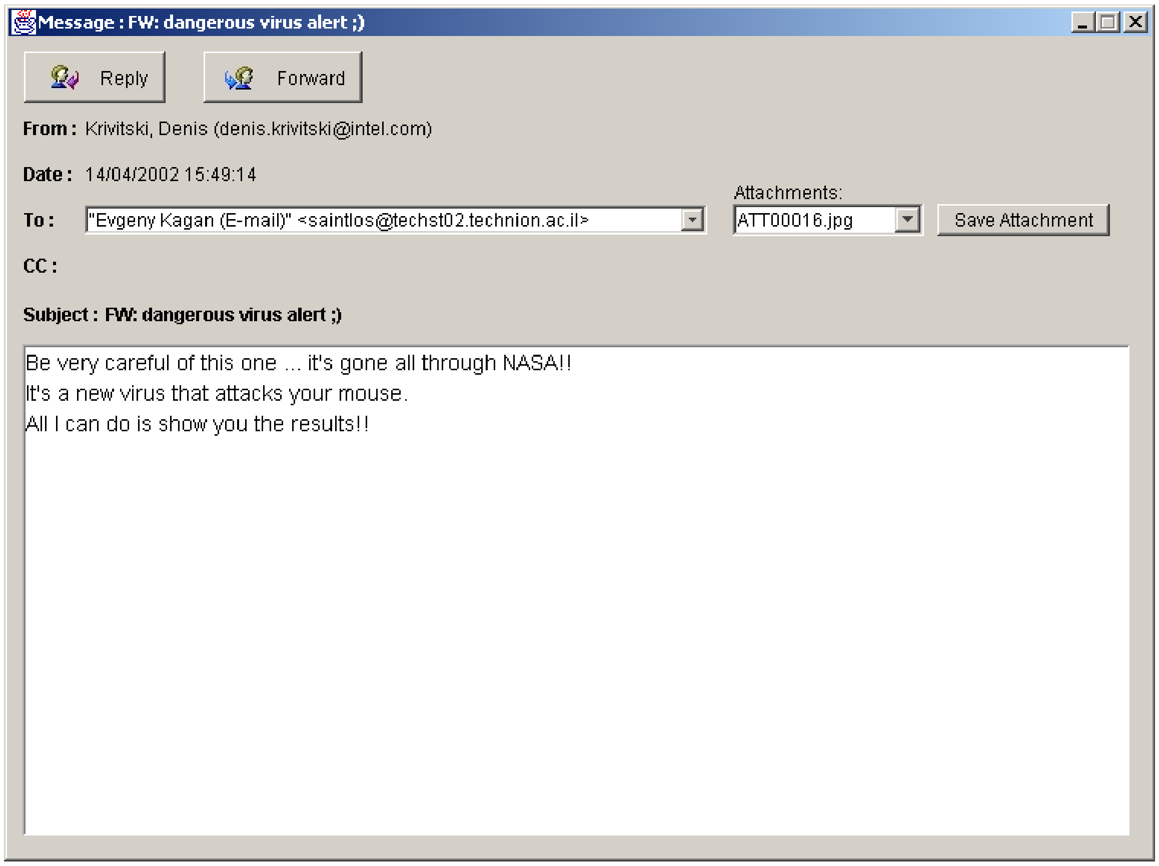
BCC :

The meeting is on : 06/02/2002 , at 22 : 06

Subject: Project review

Weekly report of the status of each groop...

1.7 *Reading Mail window* – this is a window used for reading an email message:



Message : FW: dangerous virus alert ;)

Reply | Forward

From : Krivitski, Denis (denis.krivitski@intel.com)

Date : 14/04/2002 15:49:14

To : "Evgeny Kagan (E-mail)" <saintlos@techst02.technion.ac.il>

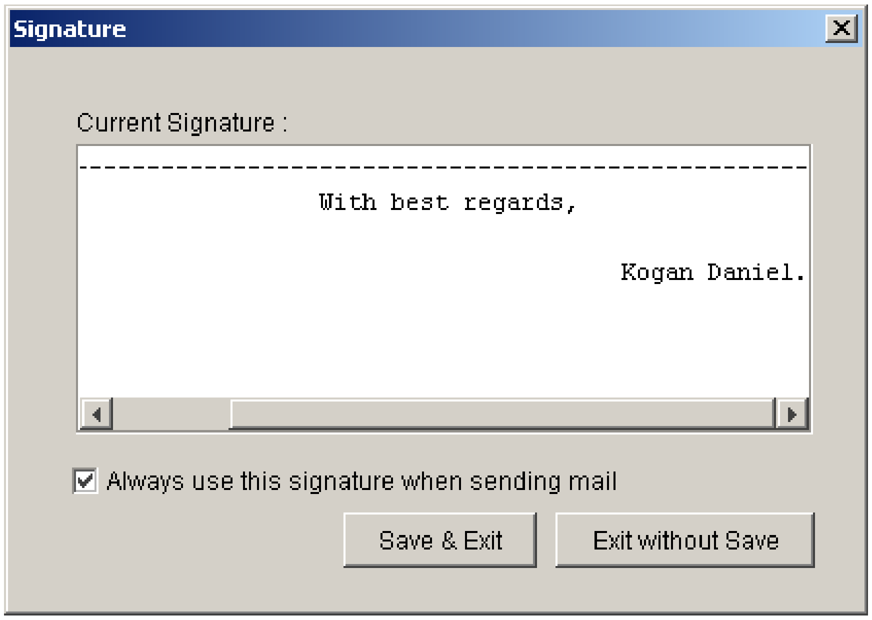
Attachments: ATT00016.jpg | Save Attachment

CC :

Subject : FW: dangerous virus alert ;)

Be very careful of this one ... it's gone all through NASA!!
It's a new virus that attacks your mouse.
All I can do is show you the results!!

1.8 *Signature window* – used for entering a personal signature of the user:



Signature

Current Signature :

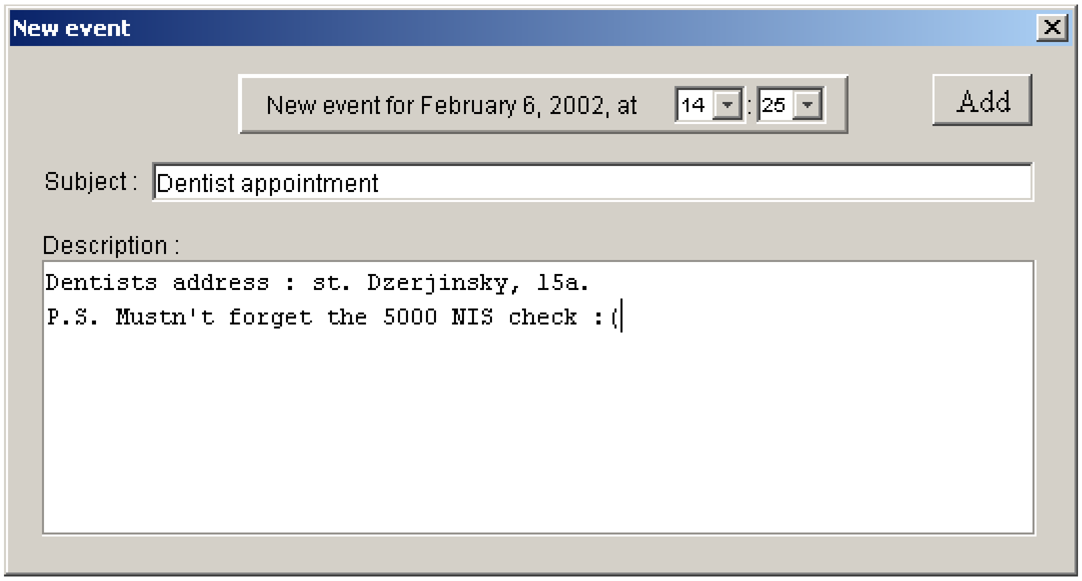
```
--------------------------------------------------
                  With best regards,

                                       Kogan Daniel.
```

[✓] Always use this signature when sending mail

Save & Exit | Exit without Save

1.9 *Event information window* – used for adding a new event to the system, and viewing an existing one:



New event for February 6, 2002, at 14 : 25

Subject : Dentist appointment

Description :

```
Dentists address : st. Dzerjinsky, 15a.
P.S. Mustn't forget the 5000 NIS check :(
```

2. Detailed instructions

2.1. Configuration

First of all, you must configure the application. The configuration will be saved to a file, and restored when you run the application the next time.

1) Open the “Configuration window” (Options-> Configuration).
2) In the “Pop3 (incoming) server” field you must specify the name of the POP3 server on which your mailbox is located (your mailbox provider must give you that information). For example, if your mailbox is at Yahoo! Mail, you need to write: “pop.mail.yahoo.com” or “216.136.173.10”.
3) In the “User Name” field you must specify you username on your mail server.
4) In the “Password” field you must specify your correct password. As you type, stars (*****) will appear for security reasons. Check that Caps Lock is not “on”. You may check the “Save password” option if you want the application to remember your password in the future.
5) If you want your recipients to see your correct e-mail address when they get your mail, you need to specify this address in the “E-mail address” field.
6) In the “SMTP (outgoing) server” field you must specify the name of the SMTP server by which you want your outgoing mail to be transferred (your mailbox provider must give you that information). For example, if your mailbox is at Yahoo! Mail, you need to write: “smtp.mail.yahoo.com” or “216.136.173.12”.
7) Please, notice the “Don't delete mail from the server” option. This is a very important option. When checked, the application will leave all the messages on the server after it gets them to the local workstation. It is useful in the case that the user is running the application on station A, and he wants to be able to get the same messages later on a machine B. However, one must remember, that most of the mail servers set a specific quota on a mailbox size, and if the mail is not periodically removed from the mailbox, it will eventually reach its limit… When the option is unchecked, all the messages are removed from the server once they are downloaded to the client’s computer.
8) In the “Scheduler options”, you may specify how many minutes before some event you want to get a reminding message.

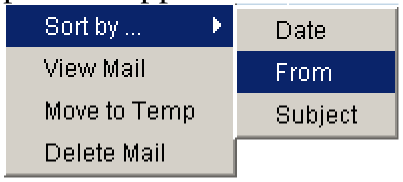
2.2. Folders

* General. In order to view the lists of messages you need to press the "E-Mail" toggle-button in the "Main" window and thus switch to the "Main window in E-Mail State". Now you can see three folders containing your mail messages:
Folder In. All the newly received mail is inserted into this folder. Later, you may delete some of the messages in it, or move them to another folder – Temp.
Folder Out. Any mail that is sent by you is automatically stored into the Out folder. Again, later you can delete it from the folder.
Folder Temp. This is a general-use folder. For example, you may want to move some messages to this folder from the In folder if they are old and do not belong with the recent messages of the In folder, but you don't want them to be permanently deleted as well.

* Pop-up menu. All the operations on folder described below are done using the pop-up menu. To invoke that menu you need to:

- Left-click (select) any message in the folder.
- Right-click on the same message.

As a result, a popup menu appears :



(This menu is slightly different in different folders).

* Sorting messages. The messages in the folders are presented in the following way – each line represents a single message and it is divided into three fields : subject of the message, its sending date and its sender name (in Out folder, instead of Sender name, the name of a first recipient appears). You can sort the messages in the folder by either one of those fields by invoking the pop-up menu and choosing the desired sorting field.
* Opening message. To open a specific message for view you can either double-click it, or open its popup menu and choose "View mail". Notice: once a mail has been opened for viewing – it is marked as "read message", and will appear in the folder list in gray color, instead of black.

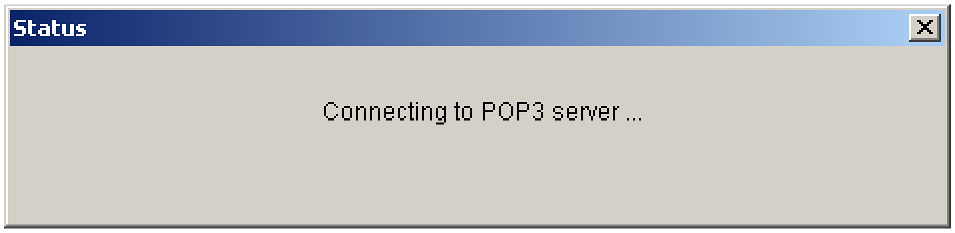
* Deleting message. In order to permanently delete some message, open its popup menu and choose "Delete". Another option is to press the DEL key when that message is selected. This will permanently delete that message if

you are in Out or Temp folder, but if you are in the In folder, this will move the message to Temp folder. (You can also move a message from In to Temp by opening its pop-up message and choosing "Move to Temp").

2.3. Receiving new mail

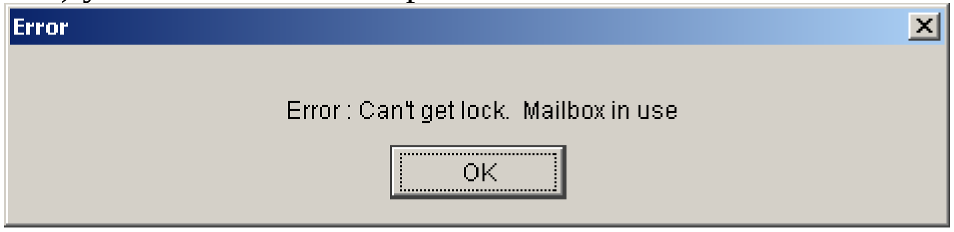
This section discusses the process of receiving messages from a remote mail server into the application's database.

* Getting new mail to your computer. To start actually receiving your mail, click on the "Get Mail" button in he "Main Window". A status window will appear:



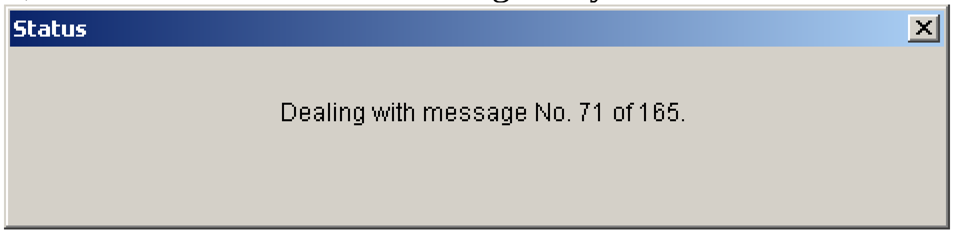
Connecting to POP3 server ...

This window will show you, on which stage of the receiving process you are. If there is some problem (wrong username / password / server, mailbox in use, etc.) you will see its description in the status window:



Error : Can't get lock. Mailbox in use

After all initializations are successfully completed, the application will start going over your remote mailbox (mail by mail) and will fetch only the "new" mail (mail that is not already present in folders In or Temp). In the status window you may see the number of the mail that is currently being checked, and the total number of messages in your remote mailbox:



Dealing with message No. 71 of 165.

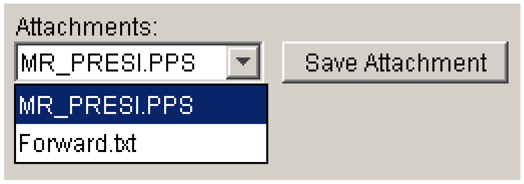
Notice, that it takes much longer to download message with a large attachment than a simple message with a text body only.

Once the receiving process is complete, all the messages will be added to your In folder's list.

2.4. Viewing the mail

* General. In this section we discuss primarily the features of the “Reading Mail Window”. That window is opened when you choose to open some message from one of the folders (see section 2.1.1->opening mail). Notice that several messages may be opened for view simultaneously! The window includes the information about its recipients (sometimes of different kinds – TO/CC/BCC), sender’s name and email, a subject, a text body, and attachments.

* Attachments appear as filenames in the “Attachments” Combo Box:



In order to save a specific attachment to some location on your file system, choose that attachment in the Combo Box, click on “Save Attachment” button, and choose the desired location in the Save Dialog that appears.

* Reply option. This option is enabled only when the opened message belongs to In or Temp folders. Pressing on a “Reply” button will:
1) Open a “Mail Composition Window” (see chapter “Writing Mail”).
2) Fill the “To:” field with the address of the sender of the original message.
3) Fill the “Subject:” field with the prefix “RE:” and the original subject.
4) Fill the mail body with a special line of the following content : “On <date> <time> , <sender> wrote :”, and then will copy the original message body with a “>” character prefix on each line.

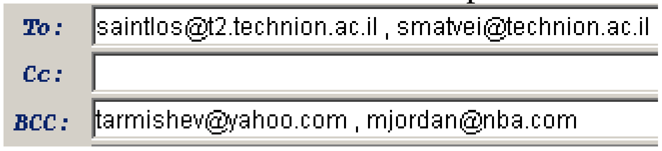
* Forward option. This option is enabled only when the opened message belongs to In or Temp folders. Pressing on a “Forward” button will:
1) Open a “Mail Composition Window” (see chapter “Writing Mail”).
2) Fill the “Subject” field with the prefix “FWD” and the original subject.
3) Fill the mail body with a special line of the following content : “On <date> <time> , <sender> wrote :”, and then will copy the original message body.
4) Add attachments of the original message to this composition window.

2.5. Writing Mail

* General. This section will explain in details how to write and send a new mail message.

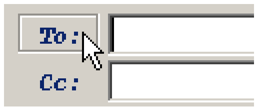
* Invocation. First, press on the “Write Mail” button in the “Main Window”. A “Mail Composition Window” will open. Notice, that you may open several composition windows concurrently.

* Specifying recipients. You **must** specify at least one recipient address in one of the fields : “To”/”Cc”/”Bcc”. You **may** specify multiple recipients in all those three fields. Separating between recipients of the same type is done by putting a comma between them. Example:

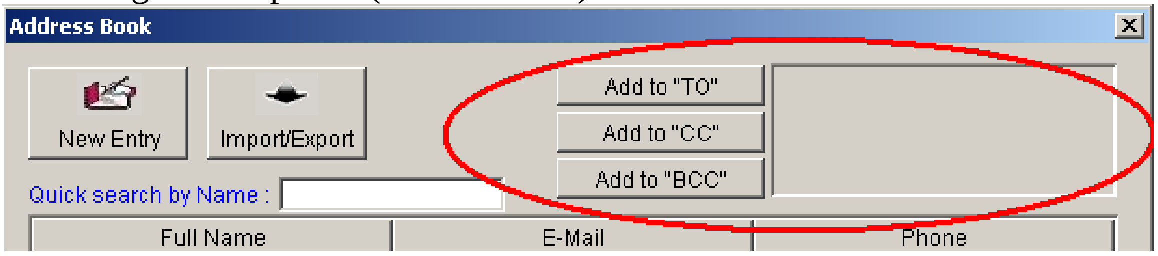


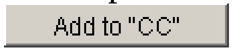
To: saintlos@t2.technion.ac.il , smatvei@technion.ac.il
Cc:
BCC: tarmishev@yahoo.com , mjordan@nba.com

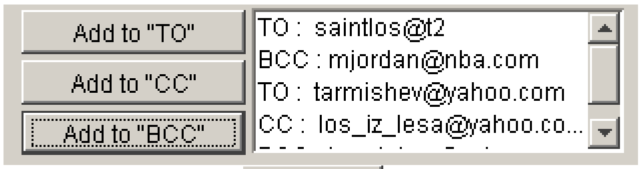
You may use the MultEMail’s address book for inserting recipients’ names. To do that you need to press on one of the buttons that are located on the “To:”, “Cc:”, “Bcc:” labels, and appear only when you put a mouse cursor on them:



This will open an “Address Book Window” with additional components for collecting the recipients (circled in red):

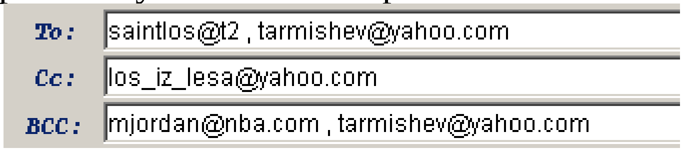


(See “Using Address Book” section for details).
Suppose you want to add some address (that exists in the address book) to your “CC:” recipients. You need to select it in the address book list, and press on  button. A new entry will be added to the recipients collection. Repeat those steps for all the addresses you need, At the end, your collection may seem something like that:



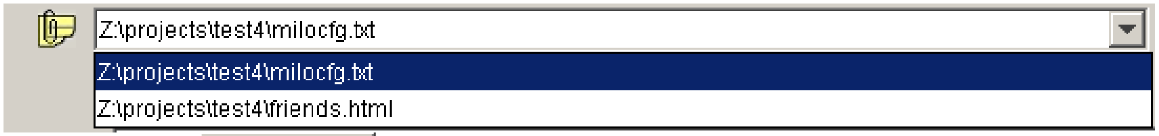
TO : saintlos@t2
BCC : mjordan@nba.com
TO : tarmishev@yahoo.com
CC : los_iz_lesa@yahoo.co...

Now, what is left is to press on OK button at the bottom of the address book window. This will close the address book and automatically add all the collected recipients to your "Mail Composition Window":



To: saintlos@t2 , tarmishev@yahoo.com
Cc: los_iz_lesa@yahoo.com
BCC: mjordan@nba.com , tarmishev@yahoo.com

* Attachments. Adding an attachment is done by pressing on the Add button, and selecting any file from the standard Open File dialog.
The selected file will be added to the attachments combo box. You may add as many attachments as you wish.
If you want to remove some attachment from the list, choose it from the attachments combo box :



Z:\projects\test4\milocfg.txt
Z:\projects\test4\milocfg.txt
Z:\projects\test4\friends.html

and press on Remove.

* Signature. The application has a feature of an automatic signature. Signature is the text that appears at the end of the messages you send, and often includes your personal and business information.
To specify and save the signature – go to Options->Signature. "Signature Window" will appear. In the "Current signature" textbox fill the signature you want. If you want this signature to be automatically added to your compositions, check the [✓] Always use this signature when sending mail checkbox. Otherwise, the signature will be saved "for better times", but won't be appended to your messages. Press on Save & Exit to save your changes.

* <u>Sending the message</u>. What is left is to fill the “Subject:” field and the body of your message. When done – you may send the message by clicking on Send It. The message will be sent and saved into the “Out” folder.

2.6. Address Book

* <u>General</u>. Mult-E-Mail manages an Address Book that enables you to add, remove, modify and easily search for address entries. It is also useful for fetching addresses while composing a new message (see 2.4 Writing Mail -> Specifying Recipients). To open an “Address Book Window”, click on “Address Book” in the main window.

* <u>Adding a new entry</u>. Click on “New Entry” button. This will open “New Address Entry Window” with all its textboxes empty. You need to fill in the information you want to be saved. You <u>must</u> fill the “Full Name:” field. Notice, that if some other entry with exactly the same full name already exists – you won’t be able to add another one, and will be asked to change that field.

Click on Add Entry to add the new entry to the database. The new entry will appear in the entries list of the “Address Book Window” (only the Full Name, E-Mail and Phone fields appear in the list).

* <u>Viewing, modifying and removing an existing entry</u>. You may view and change any information of any existing entry. Double-click on that entry in the entries list. “New Address Entry Window” will appear with all the information saved for this entry. You can change any text field you wish, and save the changes by clicking on “OK”.

To remove some entry – select it in the entries list and press on “DEL” key.

* <u>Searching the Address Book</u>. Two kinds of search are available:

- *Quick search*: this search works as following: in order to find some person in the address book (for example, “Tamara Armishev”), you should start typing his name in the Quick search by Name : field, and the application will select the entry whose name starts with the typed sequence. The problem of this search is that the you must remember what person’s name is, and how exactly you entered it into the database (if he actually entered it as “Armishev Tamara”), the quick search won’t be helpful in our example). That’s why we offer a more sophisticated type of search:

- The *advanced search*: this option allows you to find entries by many criteria: name, address, phone, and e-mail address. You may specify some of these fields, click on the "Search" button, and our program will show you all the entries that match the specified criteria (with the logical AND comparison). Moreover, you don't have to specify the full string value of the criteria, but only some part of it. Example:
The search for the criteria *Name:* "Dan", *Address:* "Haifa"
will return, for instance, entries:
Name: "Daniel Kogan", *Address*: "Neve Shanan, Haifa, Israel"
and *Name*: "Yuri Chemo<u>dan</u>ov", *Address*: "Haifa, IL",
but <u>won't</u> return *Name*: "Dany", *Address*: "Jerusalem".

2.7. Scheduler

* <u>General</u>. In order to switch to the scheduler section of the application – press on the "Scheduler" button in the main window. This will show you the "Main window in a scheduler state".

* <u>Browsing the events</u>. There are two ways by which you can look for the scheduled events:

- You may see the list of all the existing events in the "All events" list. The events are ordered by their date and time. Uncheck the "show expired events" option if you don't want expired events to appear in the list.
- To see the events scheduled for a specific date – select this date in the calendar tool and the day's events will appear in a separate list (in future referenced as a "Daily Events" list).

* <u>Viewing existing event</u>. To view the full information of some event – double-click on it in either of the two event lists described in the previous section. This will open an "Event information window" with all the event's data.

* <u>Adding a new event</u>. Suppose you want to schedule a new event for 26 of July 2079. You need to select that exact date on the calendar, and click on "Add Event" button above the "Daily Events" list. In the opened "Event information window", fill in the time of the event, its subject and body, and press on "Add".

* Removing an event. To remove an event you must select it from one of the lists (“All events” or “Daily events”, and press on the “DEL” key. The event will be erased from the database and from both lists.

* Modifying an event. You cannot change event’s data. That is because many times n event is actually a meeting request (see chapter “Meetings” below) that is sent to many users, and changing it is, therefore, meaningless.

* The reminder. In the “Scheduler options” of the configuration window, you need to specify how many minutes *T* before the event you want to get a reminding message. Once a minute, the application will go through all events, and will announce (by opening a special window and playing a sound) the events that are about to happen *T* minutes from now. In addition, when you just start the application, it will check for and announce events that are expired, but have not been announced yet (that is, the application was not running at that time).

* Meetings. One special kind of e-mail messages is a meeting. When you wish to request a meeting, you need to:
- Click on “Meeting” in the main window. That will open a “Meeting composition window”.
- Click on the button with today’s date on it. A calendar in a new window will appear. On that calendar, choose the desired date for a meeting and press on “OK”.
- Chose the time for a meeting in the appropriate combo-boxes.
- Specify all the recipients that are invited to that meeting.
- Fill in the subject and the body of the meeting.

What happens next is the following:
- A special e-mail will be sent to the recipients. They will see it as an ordinary mail with a prefix “Meeting:” in the subject. If the recipients are also using Mult-E-Mail client, then their application will recognize that mail as a meeting, and will also automatically add to the system a new event with the appropriate date, time, subject and body.
- The application of the sender (that requested the meeting) will also add the same event to its database.