EmailWebBrowser Development Guide

By

Jaswinder Bal	jbal2@uiuc.edu
Tim Brennan	tjbrenna@uiuc.edu
Kaustav Chowdury	kchowdh2@uiuc.edu
Gunther Costas	gunther.costas@gmail.com
Philip Joseph	pjoseph2@uiuc.edu

Table of Contents

Table of Figures

Table of Tables

1 Introduction

The EmailWebBrowser is an application that allows a user on a remote PC to request and then receive a web page through email. The EmailWebBrowser allows a user on a Censored Network to access a website that she is normally blocked from viewing. Censored networks can range from company intranets to a country's censored network. The EmailWebBrowser application runs on Java 6.0. The user can use any email client, however only the Opera Mail Client (minimum version 9.27) supports all existing features.

1.1 Theory of Operation

Figure 1 shows the setup of an example EmailWebBrowser system. In this diagram, the "User" emails a webpage request to the "Provider's" email address with a pre-determined subject. The Provider has the EmailWebBrowser Server installed on his/her PC and linked to an email account. When an email is received, the EmailWebBrowser Server checks the message subject for the provider-specified subject. If the message contains the specified subject, it retrieves any URLs in the message and sends the webpages back to the user.

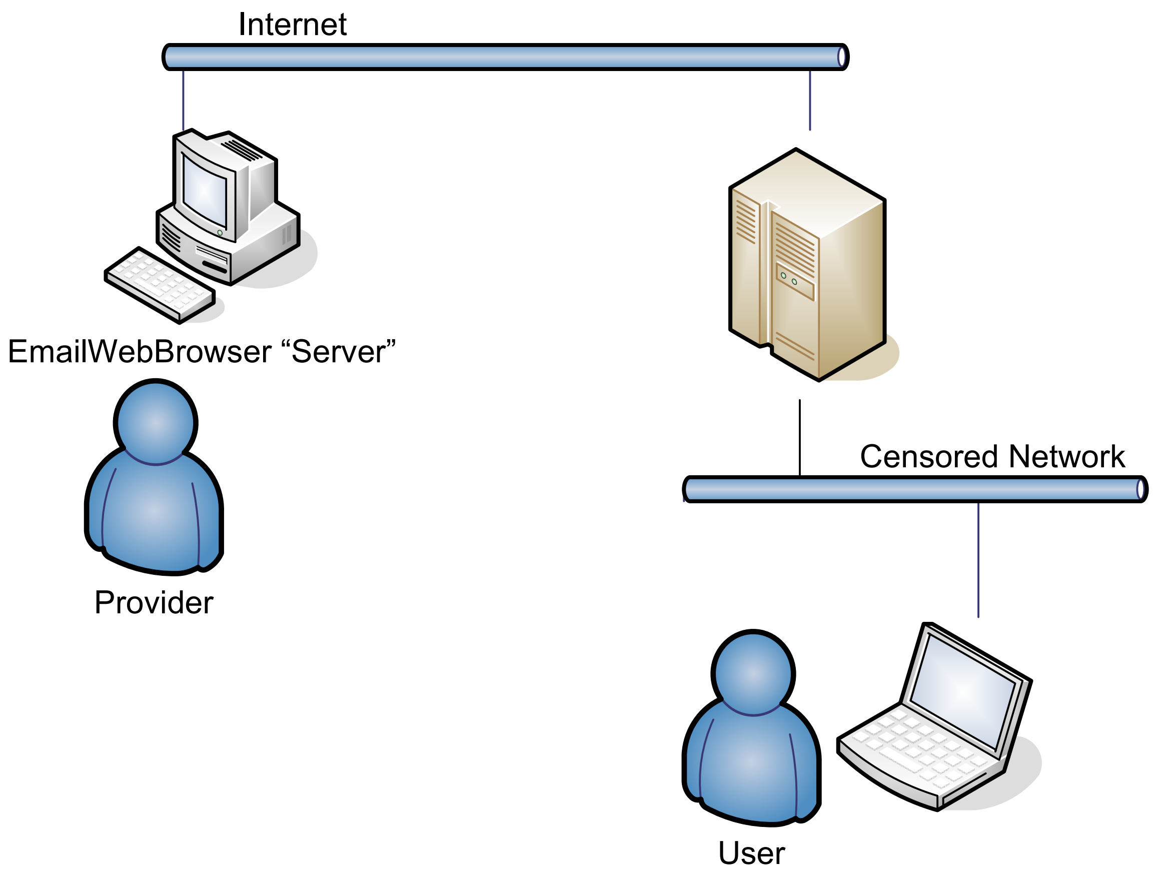


Figure 1 EmailWebBrowser System

1.2 Terminology

The terminology descriptions contained in this section along with Figure 1 above describe the terminology associated with the EmailWebBrowser System.

1.2.1 EmailWebBrowser Server

The EmailWebBrowser Server is the application that services the EmailWebBrowser requests received via email.

1.2.2 Provider

A provider is any individual that downloads, installs, and runs the EmailWebBrowser Server software.

1.2.3 User

A user is any individual that makes request to an EmailWebBrowser Server by sending an email to the email address of a known provider.

1.2.4 Censored Network

A “Censored Network” is a network or internet connection that is censored so that some webpages are blocked to users on that network; however users still have unrestricted email access. The team has personally experienced such a network while at work in a corporation. On this network, it is possible to receive email, but a large number of sites are blocked

2 Requirements

2.1 User Stories

2.1.1 User Receives Response Email

The user emails a webpage URL to a known EmailWebBrowser Server and receives a response email from the EmailWebBrowser Server.

2.1.2 User Receives Web Page Contents

The user emails webpage URLs to a known EmailWebBrowser Server and receives the webpages of the requested URLs contained within response emails. The user can view the webpages directly within his email client.

2.1.3 CSS Page Support

The user requests the URL of a webpage that uses Cascading Style Sheets to format the style elements of the webpage. The user receives a response email that contains the requested webpage with the appropriate style as specified in the CSS page.

2.1.4 Packaged as Executable

The provider downloads the EmailWebBrowser Software and installs the application. The provider then runs and configures the server with settings appropriate for his email account.

2.1.5 Hyperlink Navigation

When the user views a webpage in their email client, he clicks on a hyperlink on the webpage and is provided a draft email with the appropriate field entries to request the URL of that hyperlink. The user presses send and sometime later receives a response email with the webpage of that hyperlink.

2.1.6 HTML form entry

When the user views a webpage containing a form in their email client, he fills out the form and presses submit which displays a draft email with the appropriate field entries to request the page with the submitted form. The user presses send and then receives a response email with the page generated from the form submission.

2.2 Supported Server Platforms

The EmailWebBrowser server application runs on Java Platform, Standard Edition 6. The EmailWebBrowser server utilizes the `SystemTray` and `TrayIcon` classes within the `java.awt` package in Java SE 6. This limits the EmailWebBrowser Server to platforms that support Java SE 6 and have a System Tray (or equivalent) feature.

2.3 Supported Email Clients

While Email messages adhere to standards proposed in RFC's[1], there are no known RFC's that regulate the way CSS pages and html forms operate within an email. Some email clients limit capabilities for security and many of the web email clients such as *Gmail* and *Yahoo! Mail* limit html display capabilities in order to assure proper display of their email client. Different email clients have different capabilities; therefore, not all features are supported on all email clients. To the best of our knowledge at the time of completing this documentation, Table 1 shows the supported features of different common email clients.

Table 1 Email Client Feature Support

Email Client	Basic	CSS Pages	Forms
Gmail	x		
Yahoo	x		
Hotmail	x	x*	
Outlook	x	x	
Mozilla Thunderbird	x	x	
Opera Mail Client	x	x	x

*partial CSS support

[1] RFC – RFC stands for Request for Comment. These documents are a form of documentation of internet technologies, some of which are published as internet standards.

3 Architecture

3.1 Project Organization

The EmailWebBrowser development project is organized into packages based on functionality.

3.1.1 `configurationManager` Package

The `configurationManager` package contains the functionality that allows the provider to configure the running EmailWebBrowser server application. The package will save the settings entered to an XML file called config.xml in the EmailWebBrowser's installation directory.

3.1.2 `requestManager` Package

The `requestManager` package contains the primary functionality of the EmailWebBrowser System. It contains classes that retrieve the email from the provided account, parse the email, and retrieve the webpage. It also contains the functionality that processes the webpage, packages the webpage in an email, and sends the response to the remote user. The `requestManager` will be discussed in more detail below.

3.1.3 `requestManagerTest` Package

The `requestManagerTest` package contains automated tests for testing the functionality of the `requestManager` package.

3.1.4 `requestManagerTest.images` Package

The `requestManagerTest.images` package contains images used by the `requestManagerTest` package for testing the image handling capability of the EmailWebBrowser system.

3.1.5 `requestManagerTest.pages` Package

The `requestManagerTest.pages` package contains sample web pages used by the `requestManagerTest` package for testing.

3.1.6 `scheduleManager` Package

The `scheduleManager` package contains the functionality for launching the `requestManager` and initiating the check for new requests on the interval specified by the provider in the configuration setup.

3.1.7 `SystemTrayEWB` Package

The `SystemTrayEWB` package contains the functionality for populating a System Tray Icon on the user's desktop.

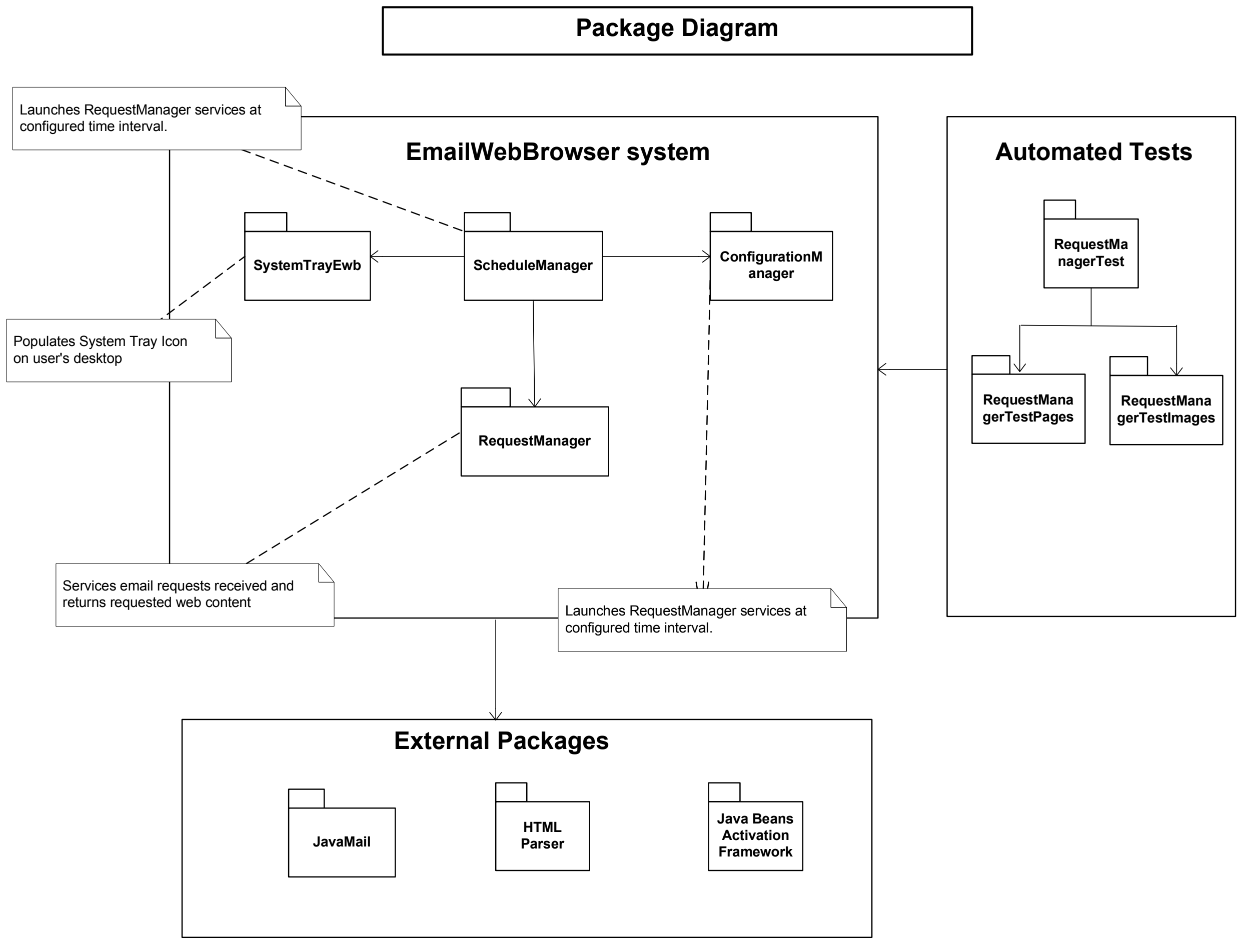

Package Diagram
Launches RequestManager services at configured time interval.
EmailWebBrowser system
Automated Tests
SystemTrayEwb
ScheduleManager
ConfigurationManager
RequestManagerTest
Populates System Tray Icon on user's desktop
RequestManager
RequestManagerTestPages
RequestManagerTestImages
Services email requests received and returns requested web content
Launches RequestManager services at configured time interval.
External Packages
JavaMail
HTML Parser
Java Beans Activation Framework

Figure 2 Project Package Structure

The EmailWebBrowser system runs as a configured service on each user's desktop in a peer to peer network architecture. This allows users to have their email requests services by other users on the internet. The main packages of system and how they rely on each other is as shown above. The EmailWebBrowser system is built making use of JavaMail and HTML Parser frameworks and relies heavily on the API provided by these packages.

3.2 Libraries Used

The EmailWebBrowser architecture is strongly influenced by the libraries used. The EmailWebBrowser uses libraries for email functionality and HTML parsing.

3.2.1 JavaMail API

The EmailWebBrowser system utilizes the JavaMail API provided by Sun Microsystems for nearly all email functionality. The JavaMail API is used to access and retrieve messages from the monitored email account. Also, the API is used to construct and send the reply email. Specifically, the EmailWebBrowser system uses JavaMail API 1.4.1 Release, available at http://java.sun.com/products/javamail/. The JavaMail API requires the JavaBeans Activation Framework (JAF) extension.

3.2.2 HTML Parser

The EmailWebBrowser system utilizes an HTML Parser to parse the webpages that have been retrieved. Parsing the webpage is required for downloading images and CSS pages for attaching to the email. The HTML Parser is also used to update URLs for the images and CSS pages to link to the `MimePart` that contains the image or CSS page. The HTML Parser is also used to rewrite hyperlinks as `mailto` links, which cause an email draft of a request for that page to be displayed when a user clicks on a link. The EmailWebBrowser system uses HTML Parser 1.6 available at http://sourceforge.net/projects/htmlparser/.

3.3 Class Structure

The EmailWebBrowser System structure can be seen in the class diagram seen in Figure 3. When the scheduler or the provider, runs the `run` method within the `Activator` class, the main email checking functionality is initiated. `EmailReader` and `EmailBodyProcessor` are the classes that process the request email, retrieve the webpage, and construct the response email. `EmailReader` utilizes `EmailImageAttacher` and `EmailCSSAttacher` to attach the images and CSS pages to the response email. The `IMapStoreConnector` Class contains the functionality to connect to the email account being monitored by the EmailWebBrowser Server.

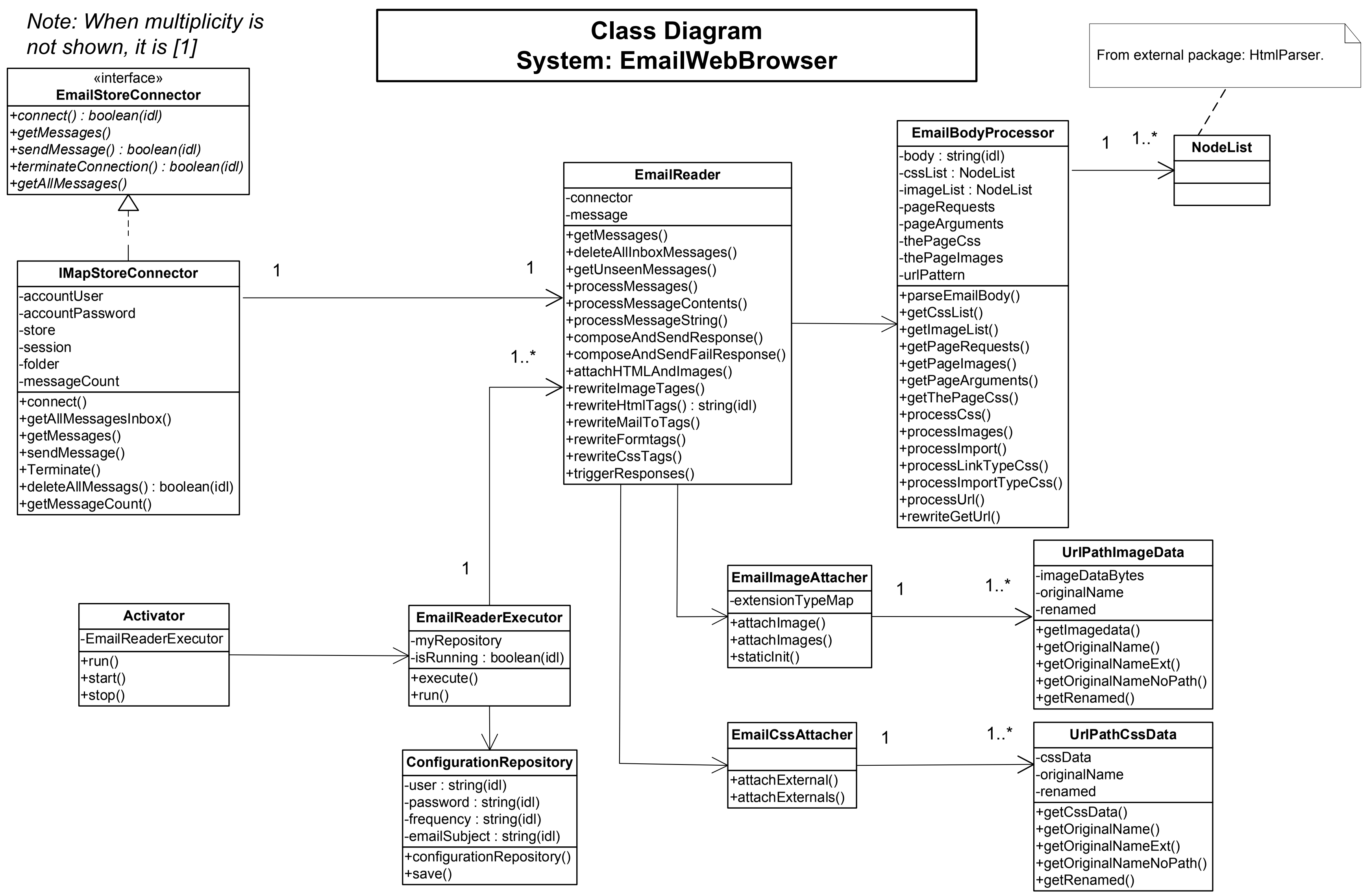


Figure 3 EmailWebBrowser Class Structure

3.4 Use Case Class Interaction

In order to understand the system architecture, it is valuable to see how classes interact to fulfill use cases. Refer to Figure 4 Sequence Diagram and Figure 5 Sequence Diagram. When the user clicks on System Setup menu Item in the System Tray the `Activator` class is initiated which invokes the `getConfiguration()` method in `ConfigurationRepository` class which accepts and saves the user configuration. The `Activator` class then calls the `execute()` method in the `EmailReaderExecutor` class which in turn invokes the `getMessages()` in the `EmailReader`. The `EmailReader` then instantiates a new `IMAPStoreConnector`, then connects to the email server and reads the messages from the registered email account. The retrieved email messages are then sent back to the `EmailReaderExecutor` class upon which it calls the `triggerResponses()` to process the email requests. The `EmailReader` then loops through each of the read email requests and gets the contents using the body content of each message `getContent()` method from the `Javax.Mail.Multipart` (part of external jar included). The `EmailReader` then instantiates the `EmailBodyProcessor` class and invokes the `parseEmailBody()` to get the email content parsed and receives a `Hashset` of the requested URLs back. For each of the requested URLs `EmailReader` then invokes `processUrl()` to fetch their respective contents. The `EmailReader` then successively calls `reWriteHtmlTags()` and `attachHTMLImages()` before invoking `sendMessage()` to send the email with the requested content over to the user who requested it.

Inside `reWriteHtmlTags()` the `EmailReader` invokes `getPageImages()`, `getThePageCss()` and then `getFullList()` to get the list of Nodes from the HTML Parser. For each of the nodes in the nodelist, it then rewrites the `mailto` tags, Image tags, CSS Tags and the Form Tags successively. Eventually The `HtmlParser.NodeList.ToHTML()` method is invoked to construct the return html page, the images are attached through the `attachImages()` method before the response is sent back to the user

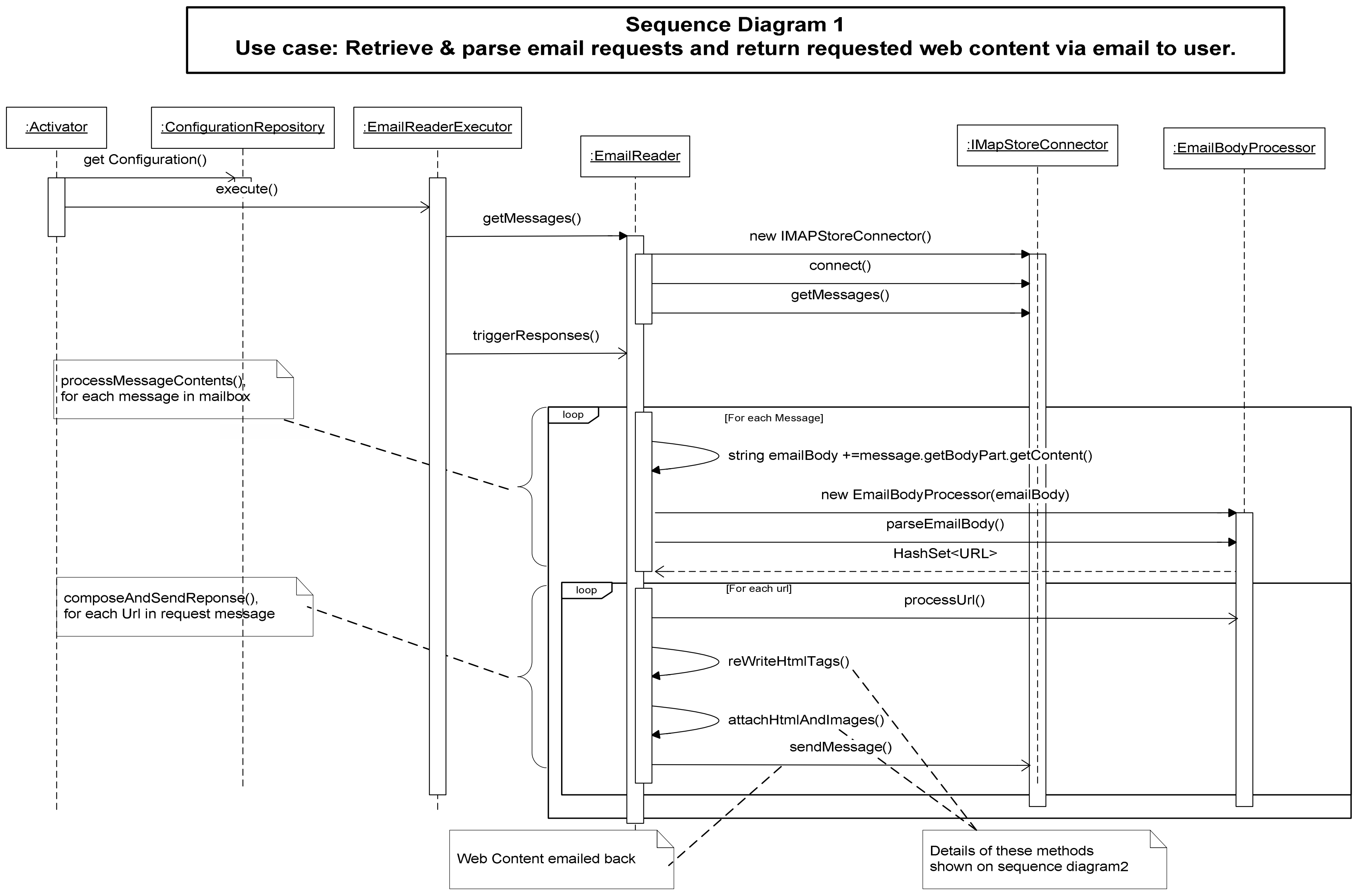


Figure 4 Sequence Diagram 1: Retrieve & parse email request and return requested web content via email to user

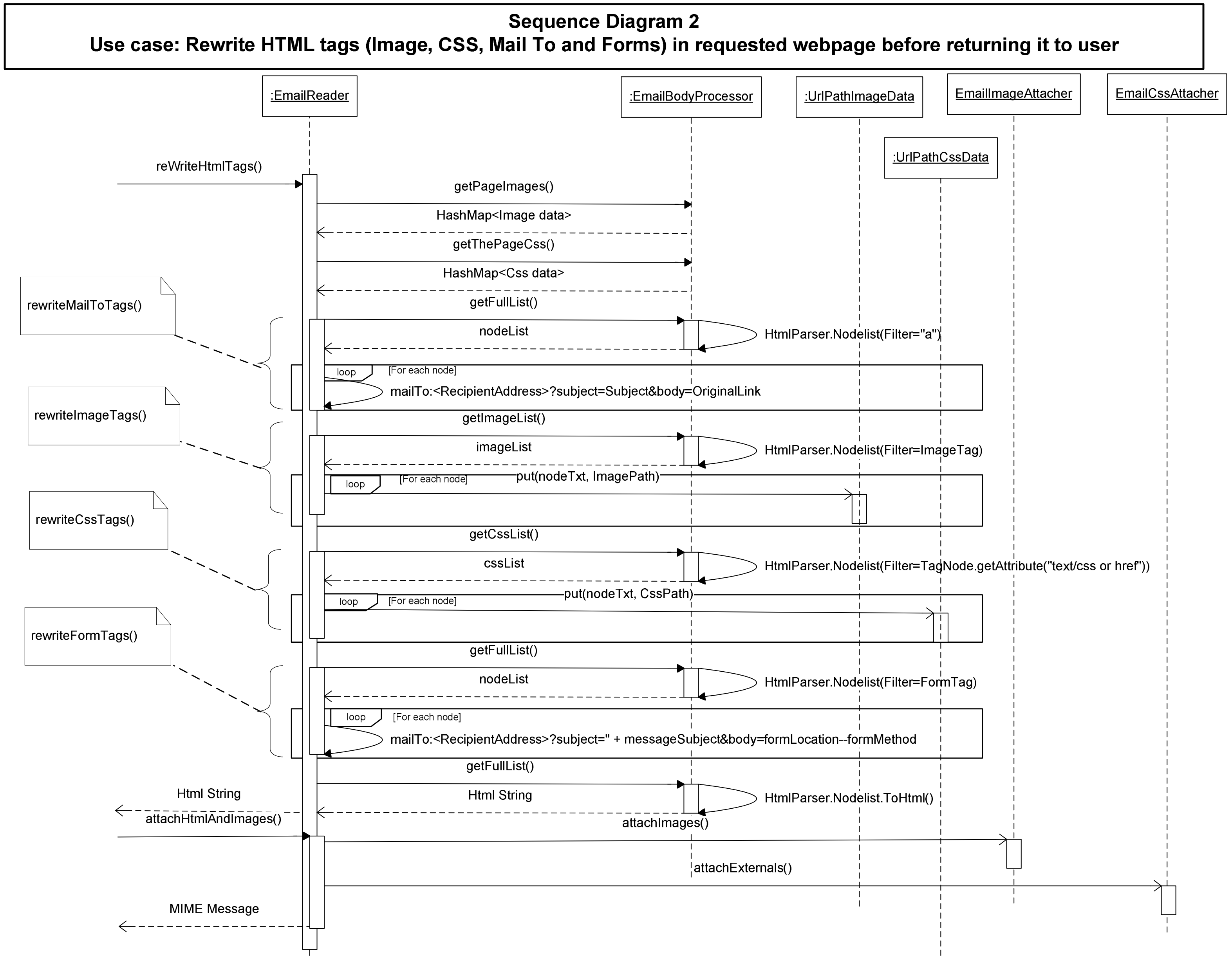


Figure 5 Sequence Diagram 2: Rewrite HTML tags (image, CSS, Mailto, and forms) in requested webpage

4 Continued Development

The EmailWebBrowser is still under development. It is currently hosted at http://code.google.com/p/emailwebbrowser/. There are a number of functions that have not been implemented that are being worked on, or will be worked on in the future.

4.1 Forms Capability

The current EmailWebBrowser version contains forms capability, however it has limitations.

The EmailWebBrowser currently supports cases of both `POST` and `GET` forms. Problems have been noted when the resulting page uses a client-side scripting language to display the items, when a scripting language is used to modify the URL that is a destination from the form, or when the user is using an email client that doesn't support emails that contain forms that use `mailto` as its `action`.

In general, the EmailWebBrowser does not correctly display webpages that are dependent on client-side scripting languages (i.e., JavaScript). In part, this is due to both offline and web email readers' inability to handle these scripts. Many forms seem to make use of these scripts to assist in displaying form results. Therefore, these forms are not fully supported.

Some webpages will use scripting to add extra information to the URL that results from clicking on a submit button. Information such as cookie information, session ID, and other client-script created information is not supported by the EmailWebBrowser.

To transfer user-inputted form data into a new email (to be emailed from the user to the provider), the EmailWebBrowser changes a form's submit button into a `POST` to a `mailto` link (i.e., `<form action='mailto: mail@email.com' method="post">`). We were able to find one offline email client that would handle this functionality correctly from within an email (most offline email clients can only handle this correctly when clicked from within a web browser context). This offline email client is the Opera Mail Client. Webmail browsers do not correctly handle this functionality. Therefore, the use of non-supported email clients (including offline and webmail) will not support the submittal of forms.

4.2 Supported Email Accounts

The current version of the EmailWebBrowser server is designed to work with email accounts through Gmail. In future versions, the EmailWebBrowser server will be configurable to work with other email services.

4.3 Supported Platforms

The current version of the EmailWebBrowser Server is only supported on windows platforms. While we are using java, part of the limitation is due to using Java SE 6 which isn't supported in all operating systems. Also the `SystemTray` and `TrayIcon` classes within Java SE 6 are only supported on operating systems with a System Tray or equivalent feature. The team's intention is to have future versions supported on multiple operating systems including Mac O/S, and some Linux distributions.

4.4 Joining Users with Providers

The current version of the EmailWebBrowser Server has no system in place to link users with potential providers. The current version relies on users to already know the email address of a provider. The team's intention is to provide some method of linking users with providers. The design of such a system has not been fully developed. An important design consideration is to make it difficult to block such a system, because users are operating on a censored network. If they cannot access the service, then it has no value.

Appendix A. License Information

EmailWebBrowser License

EmailWebBrowser.jar and all source code of the EmailWebBrowser project are licensed under GNU GPL.

Library Licenses

The EmailWebBrowser project uses several external libraries. These external libraries are licensed as shown in Table 2. Each external jar file is recognized along with the package they originated from, their license, and the directory (relative to the installation directory) that the library is installed in.

Table 2 Library Licenses

File	Package	License	Installed Directory
Activation.jar	JavaBeans Activation Framework 1.1.1.	Sun Entitlement for Software	\3rdPartyLibs\jaf\
mail.jar	JavaMail 1.4.1.	Sun Entitlement for Software	\3rdPartyLibs\javamail\
filterbuilder.jar	HTMLParser 1.6	GNU LGPL	\3rdPartyLibs\htmlparser\
htmllexer.jar	HTMLParser 1.6	GNU LGPL	\3rdPartyLibs\htmlparser\
junit.jar	HTMLParser 1.6	GNU LGPL	\3rdPartyLibs\htmlparser\
sax2.jar	HTMLParser 1.6	GNU LGPL	\3rdPartyLibs\htmlparser\
thumbelina.jar	HTMLParser 1.6	GNU LGPL	\3rdPartyLibs\htmlparser\

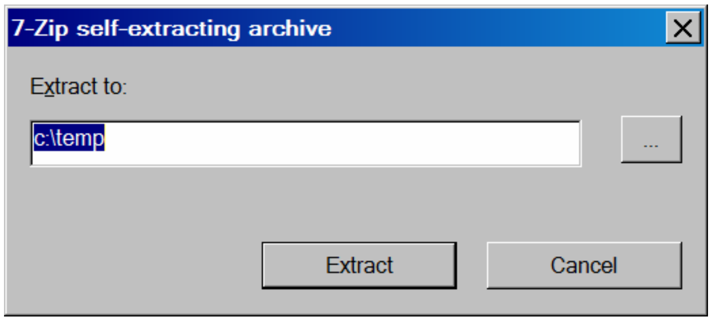
Appendix B. Installation Guide

System Requirements:

- Java SE 6
- Windows Operating System
- Gmail Account obtained free at: http://www.gmail.com

Installation

1. Download EmailWebBrowserInstall.exe from http://code.google.com/p/emailwebbrowser/. Double click EmailWebBrowserInstall.exe to begin installation
2. Specify the installation folder:



c:\temp

Figure 6 Installation Dialog

Running

1. Navigate to the location where the EmailWebBrowser was installed. Run EmailWebBrowser.bat
2. Locate the EmailWebBrowser system tray icon.

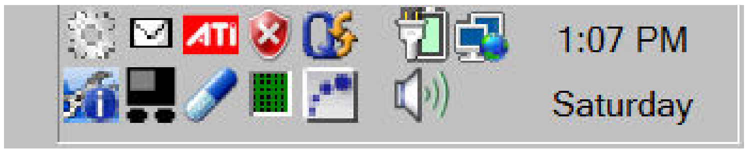


Figure 7 System Tray Icon (top left icon)

3. Right click the system tray icon and select Configure > System Setup

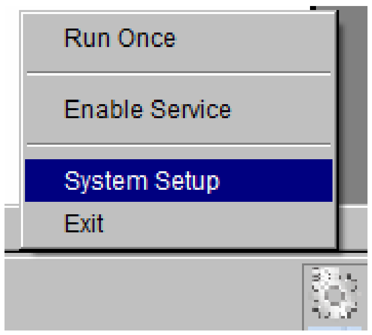
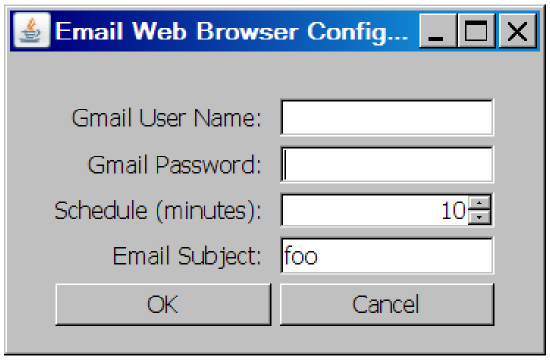


Figure 8 Menu Options

4. Enter the appropriate settings:



Gmail User Name:
Gmail Password:
Schedule (minutes): 10
Email Subject: foo

Figure 9 Settings Dialog Box

5. Right click the system tray icon and either enable the service or run the system once by selecting the appropriate selection on the menu

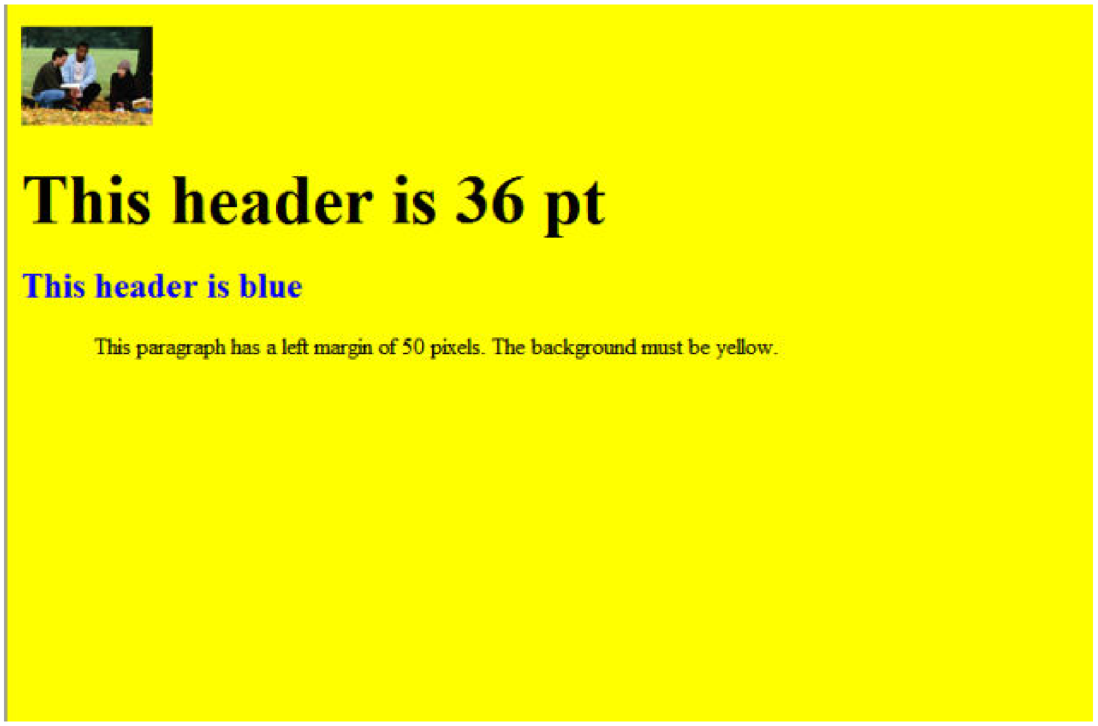
Appendix C. Manual Test Procedures

<u>CSS Pages and Web Page Displaying Manual Test Procedure</u>

<u>CSS - `Link` Method</u>

1. Have a running installation of EmailWebBrowser Server Running
2. Send an email to the email address being monitored by the EmailWebBrowser Server with the following URL:
3. http://csil-projects.cs.uiuc.edu/~tjbrenna/emailbrowsertests/csslinktest.html
4. Wait for response.
5. When response email arrives, compare returned webpage to:
6. http://csil-projects.cs.uiuc.edu/~tjbrenna/emailbrowsertests/csslinktest.html

   Verify it looks like:



This header is 36 pt

This header is blue

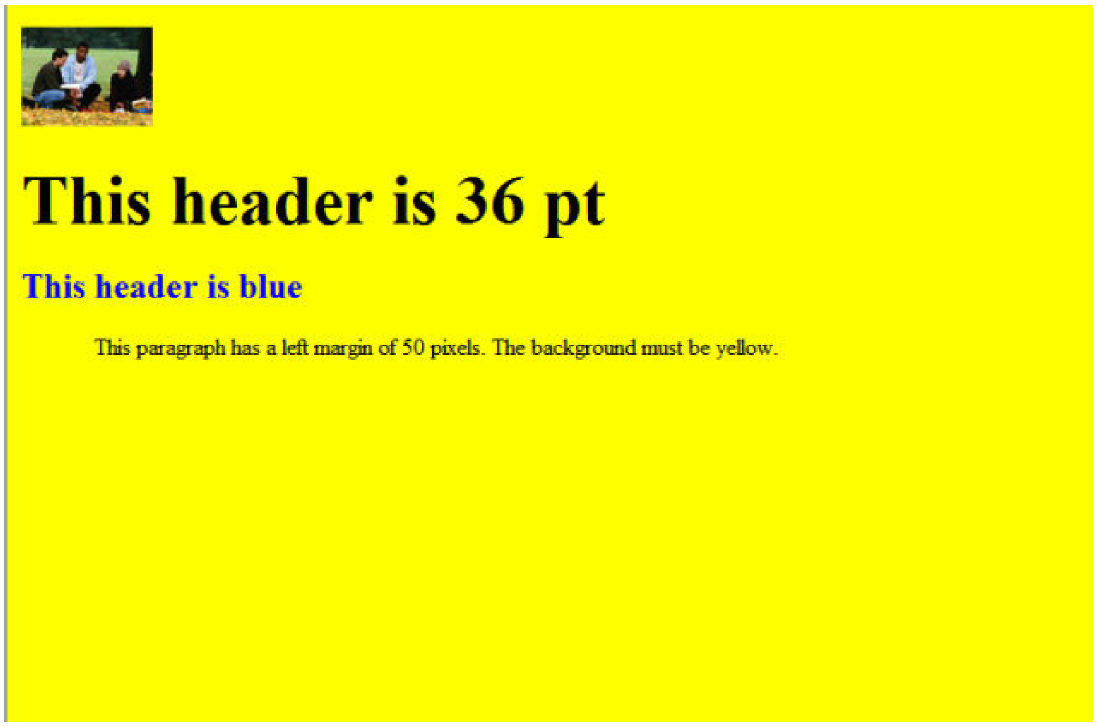
This paragraph has a left margin of 50 pixels. The background must be yellow.

Figure 10 Webpage displayed if test passes

CSS - `@import` Method

1. Have a running installation of EmailWebBrowser Server Running
2. Send an email to the email address being monitored by the EmailWebBrowser Server with the following URL:
3. http://csil-projects.cs.uiuc.edu/~tjbrenna/emailbrowsertests/cssimporttest.html
4. Wait for response.
5. When response email arrives, compare returned webpage to:
6. http://csil-projects.cs.uiuc.edu/~tjbrenna/emailbrowsertests/cssimporttest.html

   Verify it looks like:



This header is 36 pt

This header is blue

This paragraph has a left margin of 50 pixels. The background must be yellow.

Figure 11 Website displayed if CSS @import test passes

<u>Building and Installation Manual Test Procedure</u>

<u>Build</u>

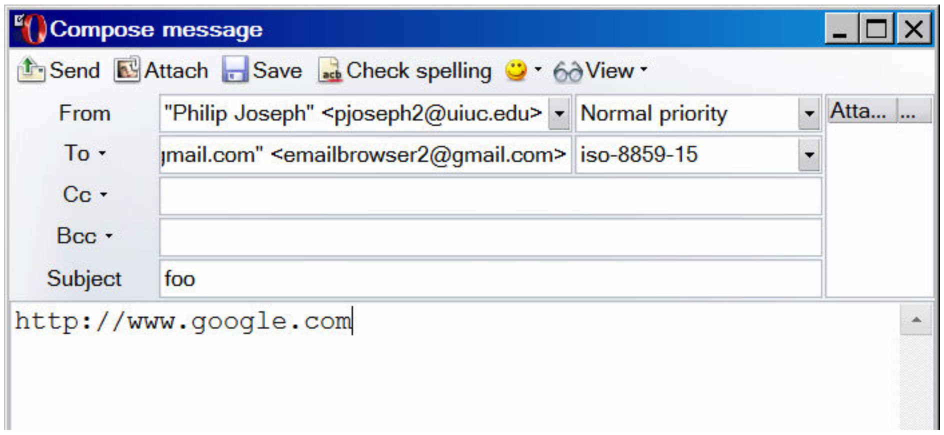
1. Navigate to the EmailWebBrowser project in the SVN repository.

2. Check out the trunk directory from 3rdPartyLibs, EmailBrowser, and Tools.

3. Verify that when going to the "Java" Perspective in Eclipse, that there are no Errors or Warnings.

<u>Installation</u>

4. In your personal email account, send an email to emailbrowser2@gmail.com with foo as the subject and a request for http://www.google.com. We will come back to this email at the end of this manual test. (Make sure to hit send/receive if necessary in order to make sure this email is sent).



From: "Philip Joseph" <pjoseph2@uiuc.edu>
Normal priority
To: ɿmail.com" <emailbrowser2@gmail.com>
iso-8859-15
Cc:
Bcc:
Subject: foo

```
http://www.google.com
```

Figure 12 Example Request Email

5. Right-Click on the EmailBrowser project and click "Refresh".

6. Verify that there is a FinalOutput folder.

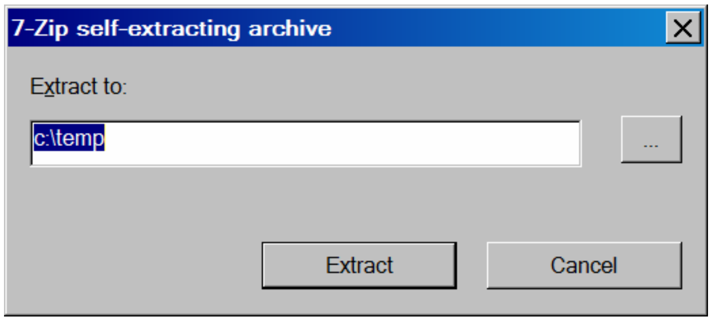
7. Verify that there are the following files in the FinalOutput folder:

    - EmailWebBrowser.jar

    - emailWebBrowserInstall.exe

8. Double-click on emailWebBrowserInstall.exe.

9. In the self-extracting archive window select a brand new folder that is separate from your workspace. Make sure to make note of this folder.



Extract to: c:\temp

Figure 13 Installation Dialog

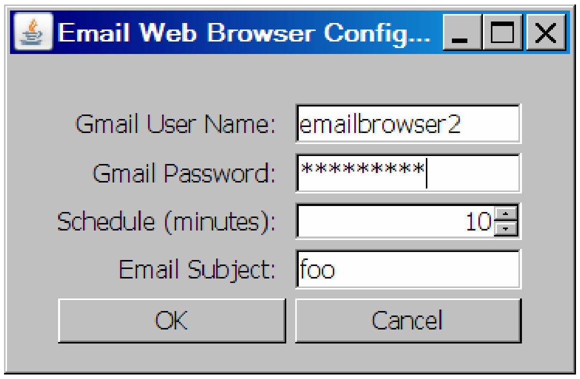
10. Navigate to the selected folder.

11. Verify that the following files and folder are there:

    - EmailWebBrowser.bat
    - EmailWebBrowser.jar
    - Gpl-3.0.txt
    - Lgpl-3.0.txt
    - 3rdPartyLibs/
    - InstallationFiles/

12. Double-click on Emailwebbrowser.bat and verify that the EmailWebBrowser icon appears in the system try.

13. Right-click on the icon, and select "System Setup", and fill in the following information (Gmail Password is: cs428pass):



Gmail User Name: emailbrowser2
Gmail Password: *********
Schedule (minutes): 10
Email Subject: foo

Figure 14 Settings Dialog

14. Right-click on the EmailWebBrowser icon and select “Run Once”. Verify that no error messages are displayed.
15. Check your personal email and verify that a response is received.

SysTrayEWB Manual Test Procedure

1) The SysTrayEWB is run as a Java Application.
   The user verifies that he can see a SysTrayEWB icon created in the system tray.

   (It is in the form of a wheel)

2) The icon is right clicked.
   Check whether The “Run Once”, “Enable Service”, “System Setup” and “Exit” menu items are displayed in the popup menu.

3) When “Run Once” is clicked it runs the EmailWebBrowser code.
   Check the program console if the results are as per expectations.

4) “Enable Service” actually enables our service and it runs according to the schedule it is set to run.
   The user should verify if he gets to see the processing of the emails happening according to the schedule that has been set.

5) The “System Setup” menu item is clicked.
   The user checks to see if it opens the configuration GUI up for the user input with Gmail username, password, schedule (minutes) and email subject input fields.

6) The “Exit” menu item is clicked.
   The user verifies that the application is exited and the system tray icon is removed.