



THE AUSTRALIAN NATIONAL UNIVERSITY

Department of Computer Science

Email Usage Statistics Collector

Yun Wang

Supervisor: Eric McCreath

June-October, 2010

Abstract

Along with the prevalence of email, a number of free email managers have been developed to assist people to deal with personal and interpersonal affairs. Beside sending and receiving email, they provide a diversity of function such as memorandum and calendar.

Many researchers are interested in how people communicate via email. They may have questions such as “how long does it take them to reply?” This project has developed and implemented an extension for Thunderbird to collect statistics about email use.

This application collects statistics from Thunderbird by which the user can simply email the result to be collected for research purpose. It is required to be compatible with IMAP protocol which is used by several email service providers like Gmail. Furthermore, it was designed to protect users' information from disclosure. To guarantee data's confidentiality, this application is equipped with an anonymiser which can replace real address with numbers. Also, it allows the users to review the statistical reports before forwarding them.

The outcome of this project is an add-on application which is able to investigate and collect data from thunderbird's embedded database and automatically compose a statistical report. It will help us to better understand humans' behaviours and improve the designs of email managers.

Recently, the add-on application has been accepted by Mozilla, and the download link is available now.

Contents

1. Introduction	6
1.1 Aims	6
1.2 Methods.....	7
1.2.1 Design.....	7
1.2.2 Implementation	7
1.3 Contributions	7
1.4 Limitations	7
1.5 Report Structure.....	8
2. Background.....	9
2.1 Concepts.....	9
2.1.1 Add-on application (plug-in application)	9
2.1.2 Embedded relational database system	9
2.2 Significance	9
2.3 Illustration of need (User stories)	10
2.3.1 In data mining	10
2.3.2 In User Interface Design.....	10
2.4 Technologies	10
2.4.1 JSON.....	10
2.4.2 XML	11
2.4.3 XUL.....	11
2.4.4 DTD.....	11
2.4.5 CSS.....	11
2.4.6 SQLite.....	11
2.4.7 Mozilla Thunderbird.....	11
2.5 Related technology	12
2.5.1 HTML and JavaScript based webpage	12
3. Requirements	13
3.1 Core Features.....	13
3.1.1 Statistical information details	13
3.2 Desirable Features (Advanced)	13
4. Schedule.....	14
4.1 Initial Plan	14

4.2 Due Dates	14
4.3 Accuracy of the Initial Plan.....	15
5 Designs & Modelling	16
5.1 System Design	16
5.1.1 User Interface Design Principles.....	16
5.3 Mozilla Interface.....	18
5.3.1 Overview of Mozilla Interface	18
5.3.2 Interface Specification	18
5.4 XUL Elements and DOM.....	19
5.4.1 Overview.....	19
5.4.2 Elements and Methods	19
5.4.3 Layout.....	21
5.4.4 Integrated User Manual	22
6. Implementation	23
6.1 UML Model.....	23
6.1.1 User Case Diagram.....	23
6.2 Database Implementation.....	24
6.2.1 Trigger	24
6.2.2 Query	24
6.3 Code & Interface Implementations.....	26
6.3.1 Database Connection Establishment &Statement Asynchronous Execution	26
6.3.2 Manipulating Listbox (Building, Emptying and Handling Click Event)	27
6.3.3 Contact Anonymisation	28
6.3.4 Folder ID & Message State Number Issues.....	28
6.3.5 Report Composing and Email review	28
6.4 Package Organization	29
6.4.1 Package Hierarchy Overview	29
6.5 License	30
7. Test & Evaluation	31
7.1 Integration Testing.....	31
7.2 Interface Testing	31
7.3 Heuristic Evaluation	31
7.4 Revision Control (Version Control)	32

8. Discussion.....	33
8.1 Conclusions	33
8.2 Issues.....	33
8.2.1 Code implementation Issue.....	33
8.2.2 Folder ID and Naming Issue	33
8.3 Future Work	34
8.3.1 Using Java.....	34
8.3.2 Applying Regular Expression	34
8.3.3 Survey.....	34
9. References.....	35
Appendix 1: Full Size Figures	36
Appendix 2: User Manual	38
Install & Uninstall Application	38
Install.....	38
Uninstall	40
Viewing Statistics	41
Customise and Send Report.....	41
Folder ID Settings.....	42

1. Introduction

E-mail has been gradually replacing ordinary letters ever since its first appearance in the 1970s. It provides a faster, cheaper and more efficient alternative to people working and studying in different fields. Along with its flourishing development, some questions proposed by human-behavioural researchers and data miners have arisen. They have been curious about people's different email reading habits and desired to collect statistics from them (e.g. similar questions were proposed in the paper named *Understanding Email Use, Conference on Human Factors in Computing Systems*, 2005[1]).

Due to massive time-consuming work, launching such a research on ordinary paper-based surveys seems impossible (e.g. in the research of *Email and Webmail Statistics*, launched by *Mark Brownlow*, 2008 [2]). Thanks to email manager and some other modern technologies, they greatly minimise the workload so that collecting statistics can be completed within seconds.

Therefore, there was an opportunity to develop a computer application which could assist those researchers to collect statistics from users' e-mail managers. These statistics will help to understand humans' behaviours.

The software was designed as an embedded add-on application of thunderbird which is a free open-sourced email manager using by millions of users. The add-on's main functions consist of:

- Collect statistics from embedded database.
- Compose statistical report.
- Forward report to a researcher.

To guarantee confidential data's safety, it also features the following:

- Anonymise users' contracts
- Allow users to review the statistical report.
- Allow users to customise the report content.

Statistics are collected from these following aspects:

- Average reply time,
- Unread e-mail,
- Deleted-unread e-mail and
- Overall statistics.

1.1 Aims

The core aim of this project was the development of the add-on application. At least, the main function "statistic collection" should be accomplished. The ultimate goal of this project was to accomplish all functions mentioned above.

Besides, it should be able to provide users a well-designed GUI dialogue using frames, windows and tables.

1.2 Methods

1.2.1 Design

This application was developed using incremental-build model where the program is accumulatively and incrementally programmed, implemented and tested. This was due to the simple-developmental environment in which no debugging tool is available.

This design approach followed top-down principles. The main function was firstly programmed and tested before the other functions were added. It was tested as a whole after each function's completion.

1.2.2 Implementation

The programming languages used in this project were JavaScript, XUL, HTML and XML. Hence, the main developmental tool should be an advanced text editor which can handle all these languages. Also, a database viewer was needed to assist debugging work.

After each function' completion, it was packed and installed into thunderbird thereby testing as a whole.

1.3 Contributions

The major contribution of this project is the comprehensive statistical data it brings to researchers. It provides an up-to-date solution which can help them to collect data without violating a user's confidentiality. This could provide researcher an insight into users' behaviours by analysing and investigating the data.

Moreover, the project's code itself provides a template or prototype for those who want to develop similar programs.

1.4 Limitations

Limitations in this project were mostly due to the short duration. A number of more sophisticated functions would be implemented if had been given more time. Moreover, the scheduled survey was cancel also due the time limitation and complex ethic-approval censor procedure.

Users sometimes need to manually set their folders IDs since they vary from provider to provider. In some cases, the setup work may be difficult for few users because some

folders' names are confusing. In case of some errors occur system may not always immediately either response to users' faulty operations or show error messages.

Other general limitations may be caused by Thunderbird, especially when users use it for the first time. Downloading email from server may cost a plenty of time when there are a host of messages to be transmitted. In extreme cases, it may take several hours.

1.5 Report Structure

The first Chapter of this report provides an outline of this project. Also, brief introductions on the developmental approaches and techniques involved in this project are listed in Section 2. Furthermore, the ultimate goal, motivation and prospective contributions are explained in the first Chapter.

In Chapter 2, techniques involved in this project are introduced by giving short definitions. Besides, basic concepts about the Thunderbird's add-on (plug-in) applications are introduced.

The requirement analysis is given in Chapter 3 followed with a timetable and arrangement in Chapter 4. Chapter 5 focuses on the techniques' details and design principles. Every technique involved in the project is elaborated in this Section.

The process of implementation is given in Chapter 6. In this Chapter, the implementation is separated into three Sections: database implementation, code & interface implementation and package organisation. A UML user case diagram is put in the beginning of this Section to illustrate the whole program's functionality.

Chapter 7 and Chapter 8 elaborate the test process and expect future works.

2. Background

This Section will give a brief introduction to the technologies and concepts involved in this project. Also, project's motivation and significances will be also depicted. Besides, each technology will be given a brief introduction.

2.1 Concepts

2.1.1 Add-on application (plug-in application)

An add-on application is a kind of software components which can be added to specific grander computer applications. Add-on applications diversify the functionalities of the pieces of software they are attached to. In this project, the add-on program enables Thunderbird to provide detailed statistics and summaries about email.

2.1.2 Embedded relational database system

Embedded database system is a tight integrated database system which is built into another application. Embedded database systems are usually dedicate to specific computer applications. Different from other database systems, they neither requires user name nor pass word to access data. Thus their response speed is much faster than the other database systems. They are usually hidden from the user interface and hardly require any maintenance. The embedded database used in this project is SQLite which the add-on application mainly interacts with.

2.2 Significance

The add-on application (email state collector) is devoted to data miners and human behavioural researchers. It enables them to collect email usage statistics from remote machines. The comprehensive statistical data it collects and compiles would be helpful in human interface design as well as software improvement.

Moreover, the patterns and features of junk mail can be found through further data analysis. After data-pre-processing, it could be used to train a neural network which can identify spam. In this way, a new approach of email classification could be established based on the rules it generates.

Besides, it can be also applied in non-computational fields, especially in psychology. By investigating different patterns of responses, some principles about people's interpersonal communication rules could be found out.

2.3 Illustration of need (User stories)

2.3.1 In data mining

The most typical potential users of this application would be data miners. Collecting data from email used to be a difficult task due to its confidentiality and other privacy issues. Email users are afraid of personal information disclosure and prefer not to give it to any institutes or companies (This phenomenon was described in the paper named *An Evaluation of Identity-Sharing Behavior in Social Network Communities*, written by *Frederic Stutzman* [3]).

In fact, many users' anxieties are mainly because of the opaque data collecting process and the misuse of data. In many cases, some researchers carelessly disclose those pieces of personal information. To be worse, some companies even exchange them for money if they think it is profitable (e.g. *Facebook Plans to Make Money by Selling Your Data* [4]).

To ease those anxieties, a transparent collecting process and an anonymising method were needed to be devised. This add-on application can perfectly handle the two difficult problems. It allows the users to review and modified the statistical report before forwarding it to researchers. Moreover, its integrated anonymiser can prevent information from re-identification.

2.3.2 In User Interface Design

Email managers vary in layouts and functions. Designers need to collect feedbacks to improve their designs. They need to determine which components are frequently used and then change the layout. For instance, buttons that frequently used should be placed on the main frame, vice versa.

By analysing the statistics, designers can easily determine which components are frequently used by users (This idea was explained in the paper named: *Used Electronic Mail to Conduct Survey Research* [5]). Moreover, it is faster and more economical than paper-based surveys because it only requires users to compose and forward their feedbacks via email.

2.4 Technologies

2.4.1 JSON

JSON (JavaScript Object Notation) was proposed and created by Bob Ippolito in 2005. It was first known as JSONP which stands for JavaScript Object Notation Proposal. It has been used continuously in webpage programming. [6]

JSON is JavaScript based programming language which is designed for data structure representing. In spite of its deep JavaScript background, it is a different open-standard and

system-independent programming language. It is an important object-oriented solution in JavaScript programming.

2.4.2 XML

Extensible Mark-up Language (XML) is set of encoding rules in web programming and other fields. It was proposed and defined by W3C in 1996 and has been developed into a number of XML-based languages. It is widely used in arbitrary-data-structure definition because of its strong support of different languages via Unicode. [8]

So far, a number of software manufactures such as Open Office have adopted its standards and applied it in their products.

2.4.3 XUL

XML User interface Language (XUL) is a mark-up language which is based on XML and many other web standards and technologies such as DTD, CSS and JavaScript. XUL was developed by Mozilla foundation and has been applied in their products. Though its name is similar to XML, it has a closer relationship with HTML since its basic syntax and structure are derived from HTML. Thus many HTML elements, attributes events can be used in XUL. [7]

2.4.4 DTD

DTD is short for Document Type Definition. It is a set of mark-up languages' definitions and declarations. It describes the elements and their attributes of a document. [8]

2.4.5 CSS

CSS (Cascading Style Sheet) is widely used for mark-up language documents description. It is not a programming language but a language style sheet which constrains elements' attributes. It has been widely applied to XML-based languages including HTML, XHTML and DHTML. [8]

2.4.6 SQLite

Pronunciation: / S·Q·Lite/. SQLite is a compact and integrated database management system which is developed using C and C++ languages. Because of its small size (around 275 kilo bytes), it is widely used in a number of small software such as thunderbird and many other email managers. [9]

Users need to download plug-in query terminals provided by others companies since SQLite provides no data query and analysis service.

2.4.7 Mozilla Thunderbird

Mozilla Thunderbird is a free-open-sourced and system-independent email manager developed by Mozilla Corporation. It provides two basic functions: email management and contact management. Since its first appearance, Mozilla has been encouraging software developers to be involved to their add-on project development. [7]

Add-on (plug-in) applications vary in functionalities, from mail searching to directory management. There are currently thousands of add-on applications available on Mozilla's web server.

2.5 Related technology

2.5.1 HTML and JavaScript based webpage

A number of web pages are developed using HTML and JavaScript which is also known as DHTML. Components such as button and dropdown list are usually linked to JavaScript functions thereby manipulating web page context. DTD files are used to define document types and CSS files control pages' layouts.

Similar, such a mechanism has been also introduced to thunderbird's add-on application development. However, they have many differences. Instead of using HTML and standard JavaScript, it uses XUL and JSON (More technical details will be discussed in the next Sections). So, this project requires a deep-background of DHTML development.

3. Requirements

In this Chapter, the requirement analysis of the project will be given. Not only the basic features, but also the desirable features will be listed.

3.1 Core Features

The core functionality of the Email statistics collector consists of following features:

- Ability to interact with the embedded database system
- Collect statistical information from different folders
 - ◆ Unread email statistics
 - ◆ Deleted email statistics
 - ◆ Replied email statistics
 - ◆ Overall email statistics
- Calculate the desired information and display it
- Compatible with IMAP protocol

3.1.1 Statistical information details

The diagram below illustrates the required information it needs to collect. It specifies information types for each required function.

	Unique ID	Authors' addresses	amount	Time interval
Replied Email	✓	✓		✓
Unread Email	✓	✓	✓	
Deleted Email	✓	✓	✓	
Overall	✓	✓	✓	

3.2 Desirable Features (Advanced)

The following features were to be added to the system as the time frame permitted:

- GUI interface
- Ability to compose reports and forward to specific email address
- Encode email addresses in report
- Authorise users to modify folder ID
- Ability to manipulate multiple email accounts

4. Schedule

This Chapter shows the initial time table of the project. The plan was slightly changed because the survey was cancelled. Due to the complexity of ethic approval process (usually need to wait for 2-3 weeks), the survey part was cancelled.

4.1 Initial Plan

Tasks	week	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
project Supervisor Selection																
Contract Finalised																
Requirements Determination																
Initial Report & Presentation																
Project Report																
Prototyping																
Database Design/Modelling																
JavaScript Code Design/Modeling																
Interface Design																
Prototype & Report Outline																
Database Implementation																
JavaScript Code Implementation																
Interface Implementation																
Test and Evaluation																
Survey																
Final Presentation																
Final Submission																
Tasks	week	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

4.2 Due Dates

Week 4: Initial presentation --- give an overview of the project and a project plan timetable

Week 9: Mid-term Project Result --- submit prototype and report outline

Week 13: Final Presentation --- includes demonstration where applicable

Projects will be due on Saturday of the 13th week (5 am, 30th October). Deliverables includes report, artefact, and final presentation slides will be submitted by that date.

4.3 Accuracy of the Initial Plan

This project was guided by the initial plan, but the plan was changed because there was not enough time to conduct a survey. Thus it was cancelled. The rest part of the plan was mostly consistent with the work progress. This time table was made in the beginning of this semester. So, it approximately scheduled an estimated project progress. Nevertheless, most tasks were finished by their deadlines. Moreover, because of the extra two weeks of the semester break; some tasks were finished earlier than scheduled dates.

5 Designs & Modelling

This Chapter focuses on the how the system was designed and elaborates the reasons and principles. This Chapter consists of four sections: *System design, database structure, Mozilla interface and XUL components*.

5.1 System Design

The design of the system follows the top-down design principles where the system's basic functions were first outlined. Then those basic elements are specified and pieced together to form a grander system. This is due to the developmental environment which has no debugging tool, and each component had to be tested in isolation.

5.1.1 User Interface Design Principles

The system follows the design principles proposed in *Design of Everyday Things* (Written by Donald A Norman; 1999) [10]. The basic principles are listed as follows:

- **Conceptual model:**

Conceptual model is the model that represents the workflow of a system. Ideally, the user's conceptual model should be equivalent to the one in the designers' mind. In this way, the user could easily get used to the system.

- **Affordance:**

It means system's usage should be recognized rather than memorised. Users tend to memorise everything's usage in daily lives. However, there are around 33,000 common things being used in our lives. Even if we only spend three minutes learning each of them, it will cost us several years. Therefore, the design should have the ability of reminding the user of its usage rather than forcing them to memorise it.

- **Visibility:**

Visibility is the ability of showing the system's working state. The user should be aware of the state the system is in and be able diagnose the system when it breaks down.

- **Feedback:**

Feedback is a part of visibility. It means the system should give the user a response when the user performs an action. This principle has been widely applied to almost all products' designs. For instance, when a key on mobile phone is pressed, it gives off a beep.

5.2 Database Structure

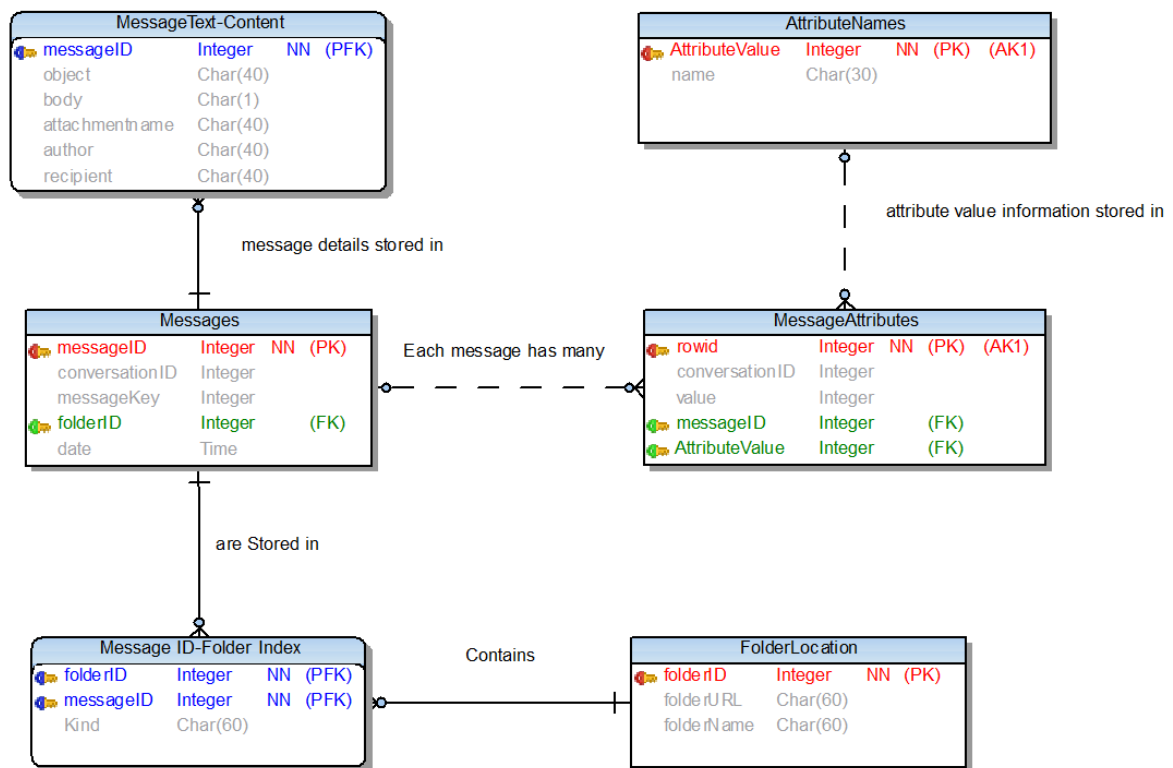


Fig. 5.1 Database Structure

In this project, there is no need to create another database or table but put modifications on them. However, this is even more difficult than creating a new one since there is no technical document or description about Thunderbird integrated database. This required a number of experiments on Thunderbird's tables' attributes' meanings and functionalities. Personally, the integrated database should be utilized and there must be a reason for why this unusual complex database structure was adopted.

Figure 1 shows the database structure of Thunderbird's build-in database. The database consists of six tables. The core table is "messages" which joins the other tables together. In thunderbird, each email is assigned to a unique ID called "messageID". Each message (email) has one or more attribute records. If any of the message's status is changed, such as it is moved to another folder or marked as read, Thunderbird will insert a new record. However, the original record(s) will not be deleted unless the email is removed.

"MessageText_Content" is a weak entity because it rests on table "Messages". Every email's details are stored in this table. Once a record in table Message is removed, its corresponding record in "MessageText_Content" will be simultaneously removed too.

5.3 Mozilla Interface

5.3.1 Overview of Mozilla Interface

Thunderbird itself was written in C++ while add-ons are developed using JavaScript, Java or JSON. In order to interact with Thunderbird, a special mechanism was developed, that is known as interface. There are more than sixty interfaces defined by Mozilla Development Community. They vary in functionalities. However, only four of them were used in this project.

Mozilla Development Community changes their interfaces 'definitions once they upgrade Thunderbird's system core. This often leads to incompatibilities. Therefore, add-on development groups often have to upload their modifications or remedies as soon as the incompatible issue arises.

Thus reducing the use of interfaces could be the best solution to prevent or reduce the incompatibility issues. All unnecessary interfaces had been replaced by sophisticated JavaScript functions in the add-on application.

5.3.2 Interface Specification

1. File service

This interface is used for file-management-related actions including file creation, searching and deleting. In the database connection phase, this interface helps to locate and specify the database file(s).

2. Storage service

Its name is somewhat misleading; this is used for database-management-related actions including connection establishment and termination, create and execute SQL statement.

3. Compose service & account management service

The two interfaces are usually combined due to thunderbird email-composing mechanism. Compose service is used for popping up a standard composing window. The events of customising email address and subject fields also rest on account management service.

5.4 XUL Elements and DOM

5.4.1 Overview

A number of components have been applied in HTML web page programming. Likewise, they are also involved in XUL programming. Though most of XUL and HTML components have a number of names in common, the usage and their methods are different in many cases.

For instance, window means browser in HTML; however, it means dialogue window in XUL. There are six different window constructors (parent, self, blank, top, media, search) in HTML while only two in XUL (with or without argument). In HTML, passing one or more parameters between windows is impossible while passing and returning parameters has been enabled in XUL.

DOM (Document Object Model) is a convention which is used for object representing and interacting. It involves document objects' attributes and methods. Similar to HTML DOM, the one in XUL adopts the same mechanism. Nonetheless, their differences lay in most methods and attributes.

For example, HTML table is merely in plain text format which means there is no action event attached to it. In contrast, Listbox (table in XUL) has a couple of action events such as "onselect()".

In conclusion, applying HTML syntax and components without any modification may cause many unpredictable errors though they are similar to some extent.

5.4.2 Elements and Methods

1. Window

There are two approaches to open a window, there were both used in the project:

(1) Open:

Syntax:

```
var windowObjectReference = window.open(strUrl, strWindowName, strWindowFeatures);
```

This constructor involves no customised argument fields and the return value is merely an indicator which shows if the window is successfully opened. In case it is not opened, the return value will be null; else the return value will be a window reference argument.

There are three arguments inside the brackets. The first one is the URL of window's XUL file. In other words, it shows where the XUL file is stored. The second argument is window's name; it can be assigned any arbitrary value. The last one, or should be call last group, indicates the window's features including window size, location etc.

(2) OpenDialog:

Syntax:

newWindow = openDialog(url, name, features, arg1, arg2, ...);

Its structure is similar to the preceding one; the difference is that it is able to pass customised argument(s) and get their return value. However, the value will not be returned until the dialogue window is closed.

2. Listbox

Listbox is used as table in XUL. As it was described in last Section, there are many methods attached to this element.

```
<listbox id="sample" rows="10" width="400">
    <listhead>
        <listheader label="1ct Gem" width="240"/>
        <listheader label="Price" width="150"/>
    </listhead>
    <listcols>
        <listcol/>
    </listcols>
    <listitem label="1"/>
    <listitem label="2"/>
    <listitem label="3" selected="true"/>
    <listitem label="4"/>
</listbox>
```

Fig.5.2 Structure of Listbox

▪ Major Methods

- Void addItem(label, value)
- Void setAttribute(label, value)
- Int getRowCount()
- Void removeItemAt(index)

Listboxes were used in displaying email information in tables. The first step is to generate blank tables with headers. They were defined in the “window” and “dia” XUL files. Next, the data from the database needed to be added into those blank tables. In order to create

cells with desired values, “setAttribute” method was used. After that, another method “appendChild” was used to add those new cells to the tree structure (table).

Listboxes sometimes need to be refreshed which means they have to be renewed. In this case, the list needs to be emptied and then filled with new elements. Emptying list in XUL is equivalent to delete all elements exist in the list. Elements (leaves) are stored in an array-like structure which supports random access.

Function `getRowCount` can get the last row’s index. And `removeItemAt` method can remove the row in specific row according to the parameter.

3. Text field, Button & Label

These elements’ usages and methods are almost the same with the ones in HTML. So, there is no need to give their descriptions here.

4. Tab

Tab elements make the window looks compact and reduce the workload. Besides, it is increasingly becoming a programming convention and being accepted as a common user-friendly design.

5.4.3 Layout

Vbox and Hbox are used to manipulate the layout in XUL. This is different to HTML. In HTML, “
” and “<p>” tags are usually used to control the layout. However, there neither “
” nor “<p>” in XUL. Instead, “<vbox>” and “< hbox>” tag were adopted to control XUL’s layout. Vbox means vertical box and Hbox means horizontal box. Elements in Vbox will be placed vertically. Likewise, elements in Hbox are horizontally placed.

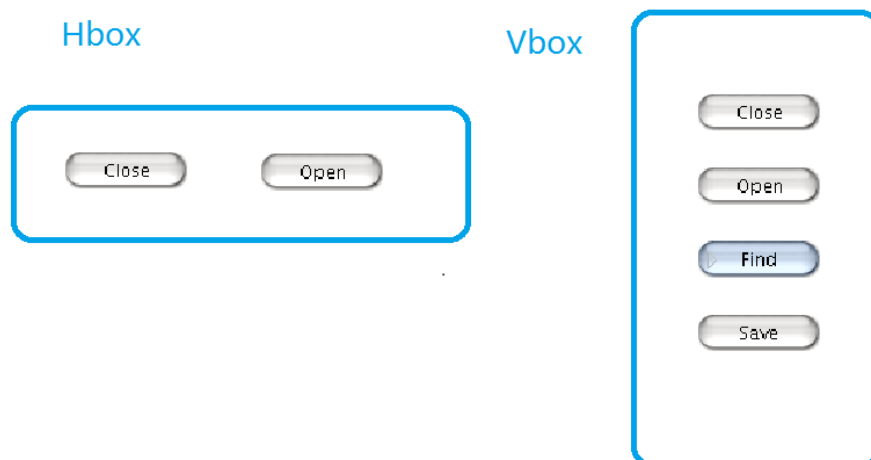


Fig. 3 Vbox and Hbox

5.4.4 Integrated User Manual

The HTML-styled user manual was integrated into the system so that the user can get help while the program is running. Compared with paper-based user manuals, the integrated ones have more preferable strong points. The user no longer needs to operate the system while holding a pile of paper. Furthermore, most software companies adopt integrated manuals rather than paper-based ones.

6. Implementation

The implementation process will be elaborated in this Chapter. Moreover, encountered difficulties and their solutions will be also explained. The first section, *UML Model*, gives an overview of this application's functionality. Then, the database implementation, code implementation and package organisation will be explained respectively.

6.1 UML Model

6.1.1 User Case Diagram

The program's user case diagram is given below:

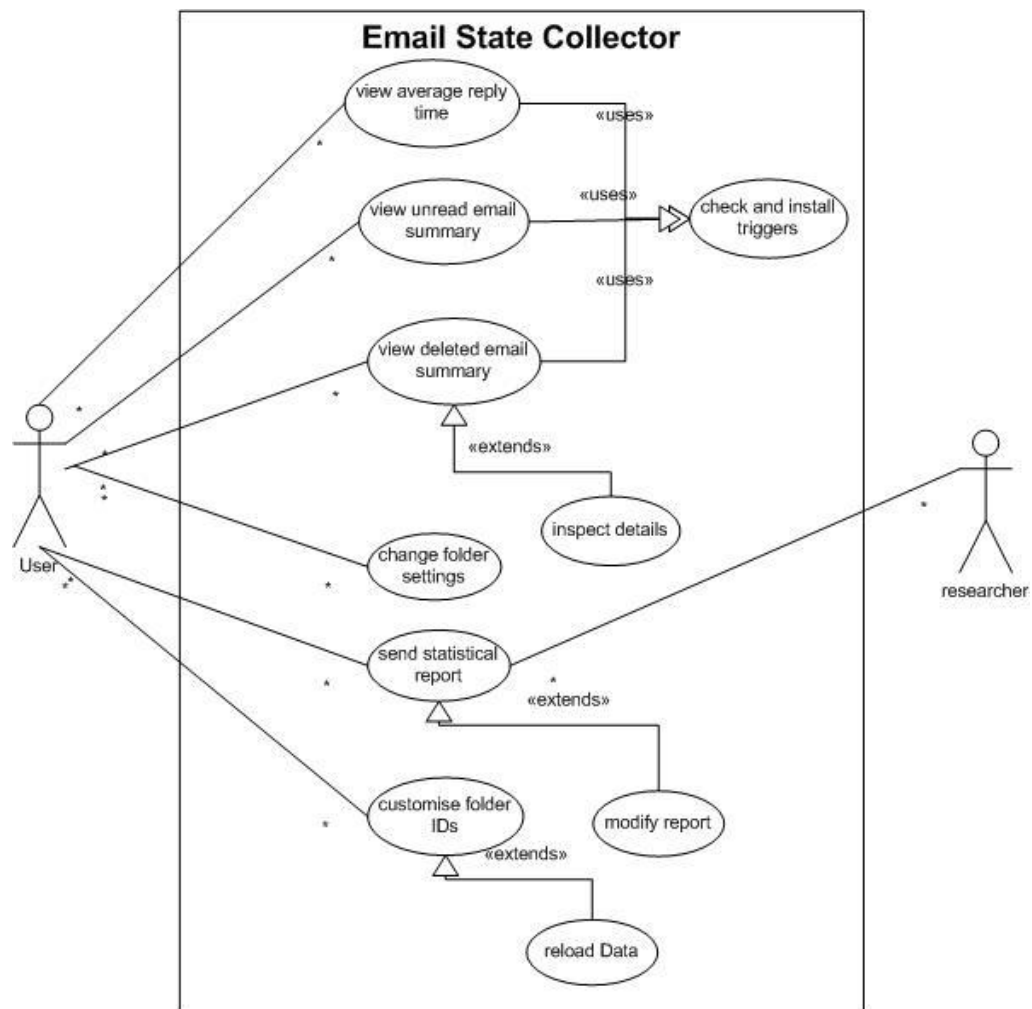


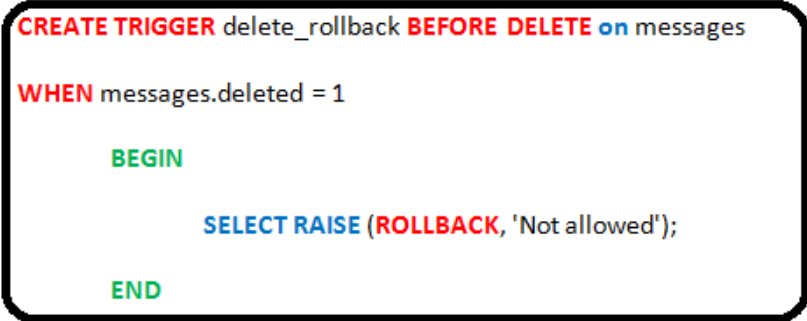
Fig.6.1 User Case Diagram

6.2 Database Implementation

6.2.1 Trigger

As elaborated in the preceding Chapter, the corresponding records will be removed when the email is deleted. Thus some modifications were needed to be added to the integral database thereby preventing data losing. There were three approaches to implement this change. The first solution is to create a new table to collect all necessary information from incoming email. However, its major drawback is that it collects no data from the email that already exists in the database. Moreover, some tables in the database should be at least utilized. The second solution is to build a view which contains several columns from the tables. The problem is that the view changes whenever the table is altered. It cannot keep all records. The last method is to add triggers on some of the tables. It is necessary to give a short introduction on trigger for it is seldom used. Trigger is a kind of SQL statement which works on other SQL statements. It was designed to response to certain SQL transactions. For instance, some triggers can be designed to automatically roll back transactions. In this case, triggers that can roll back “delete” transactions were needed to be applied to several tables.

According to Figure 5.1 in the last Section, triggers should be applied on three of those entities --- Messages, MessageAttributes and AttributeNames. There is no need to put a trigger on table FolderLocation since it has no direct relations with messages (email)’s records.



```
CREATE TRIGGER delete_rollback BEFORE DELETE on messages
WHEN messages.deleted = 1
BEGIN
    SELECT RAISE (ROLLBACK, 'Not allowed');
END
```

Fig.6.2 Trigger on table “messages”

Figure 6.2 shows a typical trigger which was applied on table “messages”. The syntax is simple. Most important of all, it should work before the delete action; otherwise records will be permanently and irreversibly lost.

6.2.2 Query

After the triggers were added, the query statements were needed to serve the main functionality. This Section focuses on how these three SQL statements were designed and the reasons behind them will be given.

Because of the complexity of the database's structure, those statements' lengths are long and difficult to understand. An alternative solution was to create a view and then query from it. Theoretically, this method can greatly reduce the query time. Nonetheless, it makes no differences on the embedded database system. In fact, it costs both of them the same time to complete an identical task. Undoubtedly, creating a view is unnecessary in this case.

```
select round(avg((second.date - first.date)/3600000000.0), 3)as averagetime,
messagesText_content.c3author as c3author from messages as first left join messages as
second on first.conversationID = second.conversationID left join messagesText_content on
first.id = messagesText_content.docid where first.conversationID in (select distinct
third.conversationID from messages as third where third.folderID in (" + sent +")) and
first.folderID = 1 and second.folderID in (" + sent +") and first.date < second.date group by
messagesText_content.c3author
```

Fig. 6.3 SQL statement of average reply query

As it shows in Figure 6.3, the complex SQL statement was applied with a number of essential SQL techniques including sub query and self join. In order to investigate the average reply time, there were several pieces of information needed to be collected. First of all, the incoming and outgoing emails had to be matched so that the time interval could be calculated. However, not all incoming email has their corresponding replies since most of us do not reply all email. Similarly, some outgoing email has no corresponding incoming email because we sometimes write to people who have never sent us any email. It seems that this work must be difficult and complex. However, thanks to the attribute of conversationID by which the recorders in table “messages” could be grouped and sorted. This problem had been simplified by the mechanism which checks if each conversation has corresponding emails both in incoming and outgoing folders, then pick up those which satisfy this condition for further processing.

Remarkably, the table “messages” was used three times in this SQL statement. Aliases were used to help to distinguish them. In this table joining step, due to the limitation of integrated database, only left join can be used (It has no join, full join or right join).

The variable “sent” was used for customising folder ID, because different email service providers vary in folder ID settings.

Finally, the result needed to be transformed to an understandable format. The date format in the database is a long type integer which represents the UNIX time (since 1st, January, 1970). There are two reasons for Thunderbird to adopt UNIX time: there is no date format in the integrated database and the UNIX time format can be easily calculated. After the subtraction, the time was divided by 3600,000,000 so that it would be represented in the unit of hour (1hour = 60 minutes = 3,600 seconds = 3600,000,000 milliseconds).

```
SELECT COUNT (*) AS amount, messagesText_content.c3author AS c3author FROM messages LEFT
JOIN messagesText_content ON messages.id = messagesText_content.docid WHERE messageKey is
NULL GROUP BY messagesText_content.c3author
```

Fig. 6.4 SQL statement of deleted email query

Compared with last SQL statement, this one is relatively simpler. One of the deleted email's remarkable features is that its messageKey value is null. Thus the amount could be calculated by counting records with null values in messageKey fields.

```
SELECT COUNT (*) AS amount, messagesText_content.c3author AS c3author FROM messages LEFT
JOIN messagesText_content ON messages.id = messagesText_content.docid WHERE
messages.conversationID IN (SELECT DISTINCT conversationID FROM messageAttributes WHERE
messageID>0 GROUP BY messageID HAVING rowid = MAX(rowid) AND value IN (" + unreadValue + "))
AND messages.folderID IN (" + inbox + ") GROUP BY messagesText_content.c3author
```

Fig. 6.5 SQL statement of unread email query

Figure 6.5 shows the SQL statement of unread email query. Some of the values such as folder ID (inbox) and unreadValue can be customized by the user.

As it was explained in Chapter 5, Thunderbird deleted no records from table “MessageAttributes” unless the corresponding email is deleted. Therefore, each piece of email may have more than one corresponding records in table “MessageAttributes”. However, only the last record of each piece of email shows its current state. The primary key of table “MessageAttributes”, rowid, can show the sequence of insertion. The latest records have greater rowid than the old ones. Those last records can be located by using function MAX. However, function MAX cannot be put in WHERE statement since it is one of the aggregation functions (the others are AVG, COUNT, MIN, and SUM). Rather, it can be put in HAVING statement.

The next step is to find out all unread email by investigating table “Messages”. Later, the two tables, “Messages” and “MessageAttributes” joint together for amount counting.

6.3 Code & Interface Implementations

6.3.1 Database Connection Establishment & Statement Asynchronous Execution

Thunderbird provides a unique way to establish a database connection. It first creates a file pointer by calling a Thunderbird interface. Then, the system uses the file pointer to point to the database file. After that, another interface will be called to establish a connection through the file pointer.

The process of reading data from database is called asynchronous execution. The execution will not get stuck despite errors. The asynchronous function handles different processes separately (figure 6.6); the last two processes are optional.

```
statement.executeAsync({
  handleResult: function(aResultSet) {
    for (let row = aResultSet.getNextRow();
        row;
        row = aResultSet.getNextRow()) {

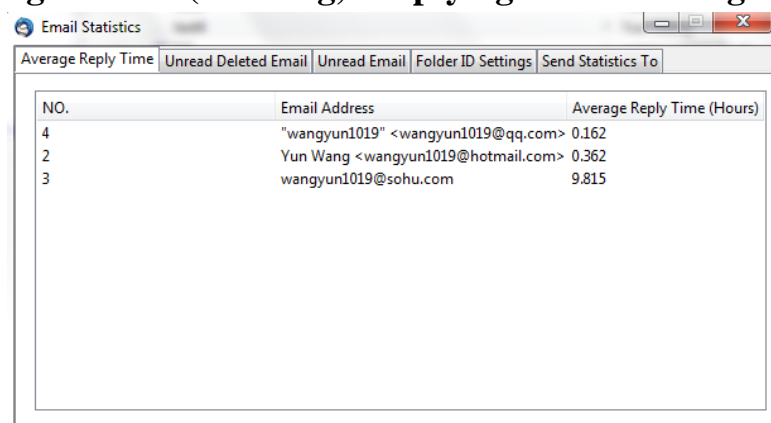
      let value = row.getResultByName("column_name");
    }
  },

  handleError: function(aError) {
    print("Error: " + aError.message);
  },

  handleCompletion: function(aReason) {
    if (aReason != Components.interfaces.mozIStorageStatementCallback.REASON_FINISHED)
      print("Query canceled or aborted!");
  }
});
```

Fig. 6.6 Asynchronous function

6.3.2 Manipulating Listbox (Building, Emptying and Handling Click Event)



The screenshot shows a window titled 'Email Statistics' with several tabs: 'Average Reply Time', 'Unread Deleted Email', 'Unread Email', 'Folder ID Settings', and 'Send Statistics To'. The 'Average Reply Time' tab is active, displaying a listbox with three rows of data. The listbox has three columns: 'NO.', 'Email Address', and 'Average Reply Time (Hours)'.

NO.	Email Address	Average Reply Time (Hours)
4	"wangyun1019" <wangyun1019@qq.com>	0.162
2	Yun Wang <wangyun1019@hotmail.com>	0.362
3	wangyun1019@sohu.com	9.815

Fig. 6.7 Appearance of Listbox

As it was explained in the preceding Section, table in XUL is called Listbox. In the beginning, several empty listboxes were defined in the XUL file. Elements will be added to those listboxes when the system reads from the database.

Like in HTML, elements in XUL files also have attributes such as ID by what the JavaScript file can locate and identify them. Building a listbox is just like adding branches and leaves to a tree since it is of tree structure.

Once the user customises the values, all those listboxes will be rebuilt. The process of Rebuilding consists of two steps, emptying and building. Thunderbird provides no methods to empty listbox. So, those listboxes have to be emptied item by item. Every listbox item has an index number; they pile up like a stack of which only the last index number is known and available. Thus, items can be removed one after another through their index numbers.

6.3.3 Contact Anonymisation

The purpose of contact anonymisation was to prevent the user's privacy from disclosure. There was no need to develop a special algorithm to code the email addresses. Rather, an array was adopted to store the addresses and their unique numbers. However, this task was not as easy as it seemed. Firstly, the address strings stored in the database usually have blank space either in the beginning or in the end. In this case, those strings needed to be pre-processed and then could be compared with each other. Regular expression was used in this step to remove the blank spaces. It turned out to be a successful approach.

The second step was to assign each distinct email-address in the database a unique number when this program started. There is neither container structure (like array list or linked list) nor class-like structure exists in JavaScript (Class does exist in JSON. However, unlike in Java and C++, the class variables in JavaScript are used as pointers to point to the class itself). Thus array became the only choice which merely allows one dimensional strings or integers to be stored. In spite of this, the array's index number could be utilised as the unique numbers for all addresses.

6.3.4 Folder ID & Message State Number Issues

All email service providers have their own system structural designs which result in differences in folder identifications. Similarly, they also adopt various states to describe email. For instance, email can be marked as 'read', 'unread', 'deleted', 'replied' in the systems using different numbers. However, different server providers usually use different numbers to represent the same states; some of them even have their own created state such as 'stared'.

In Gmail, the incoming folder's ID is one while it is seven in SOHU mail. A feasible solution is to use regular expression to automatically detect those folder IDs by their names. However, those names also vary from one to another. For instance, some service providers use 'sent items' or 'sent' while some others use 'outbox'. To be worse, some email systems have duplicate folders in different names (Gmail has both outbox and sent item).

No machines can be more intelligent than human being. So, this work can be manually done by the user within seconds. The program lists all folders and their corresponding IDs and requires the user to fill the textboxes with correct ID numbers.

6.3.5 Report Composing and Email review

Email addresses are visible to the user since it is easy for them to review their email. On the contrary, they are replaced by the unique codes in the report for the sake of privacy.

The report is an automatic composed email which contains information assembled from the listboxes. Though it is in HTML style, not all HTML elements can be applied. The recipient's email address can be customised so that it can serve different researchers.

The report is stored in a string. Unusually, the size of string type variable in JavaScript is relatively unlimited. It can be enlarged to the maximum size of the system's available memory. Therefore, hundreds of words can be stored within one string.

6.4 Package Organization

An add-on program consists of many files in different folders. Those files should be stored in a specific hierarchical structure so that they can be located by Thunderbird. Thunderbird has a number of file restrictions and definitions which can be found both on its official website and publications. In this Section, the package hierarchical structure as well as the mechanism of calling add-on application will be elaborated.

6.4.1 Package Hierarchy Overview

One of Thunderbird's official publications, XUL Tutorial, uses more than thirty pages on explaining add-on application package hierarchy since it is very essential. By no means do I want to spend so many pages on the same topic, rather I will give only a brief overview.

Typically, an add-on application includes three indispensable folders and two definition files named *“contents”*, *“locale”*, *“skin”*, *“chrome.manifest”* and *“install.rdf”*. *“Install.rdf”* gives add-on application's information summary (such as application name, ID, author name and version number). Also, URLs are also kept in this file. Thunderbird will first inspect this file when the add-on application is being installed. Some of those fields such as compatible version number should be set correctly or installation could be suspended or rejected by Thunderbird.

“Chrome.manifest” defines the root directory and the location of main method (main file). After the inspection of *“install.rdf”*, Thunderbird will scan the manifest file so that it can locate the main method (the install file specifies the main methods' name, and the manifest file tells which JavaScript file contains the main method).

Folder content contains the JavaScript files and their corresponding XUL files. However, there is a trend of using one Jar file instead of them. Those BAK files are log files where the modification histories are stored.

Folder skin was designed to store CSS files which constrain the layouts of XUL files. Usually, the JavaScript file and its corresponding XUL and CSS files share the same name so that they can be easily recognised.

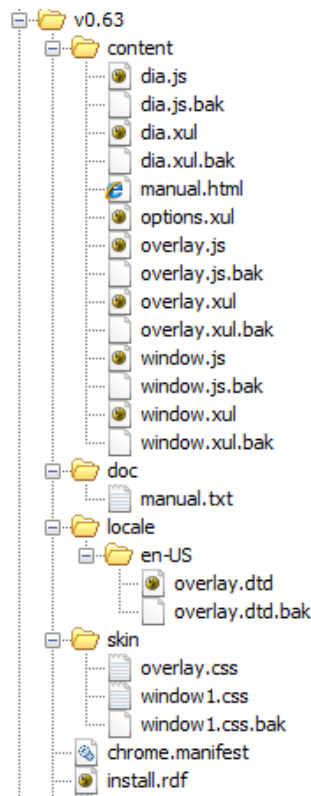


Fig. 6.8 Package Hierarchy of the Project

Similarly, DTD files are kept in folder locale which mainly contains language definition files. In this project, there is a customised folder named doc which includes user manual and help information. The package structure of this project is shown in Figure 6.8.

6.5 License

This application has been accepted by Mozilla Developer Centre, and its download link is available now. The license of this project is GPLv3 and the source code is available to the public. This kind of license allows the other programmers to contribute to the project as long as they put the acknowledgement in their source code.

Making the source code available to the public can keep this project alive and active. In other words, any programmer can be involved in this project if they want. The future work is by no means easy and perhaps cannot be accomplished within a semester. So, only in this way do I think the future work can be fulfilled.

7. Test & Evaluation

This chapter introduces the methods that used in the test and evaluation process. Also, the test result will be presented.

7.1 Integration Testing

In the software test Section, each function was test separately before they were tested as a whole. The tool used in this Section was JUnit, which is widely used in Java and JavaScript testing. JUnit has been integrated into Eclipse; thus the test was launched on Eclipse IDE platform. Table 7.1 below shows the result of each function's test.

Task Attempted	Bugs Found	Date of Test
Check and Insert Triggers	2	10th/August/2010
Average Reply Time List	4	16th/August/2010
Unread Email List	3	20th/August/2010
Deleted Email List	3	22nd/August/2010
Folder ID List	0	15th/September/2010
Edit Settings	1	20th/September/2010
User Manual	1	18th/October/2010
Compose Report	3	13th/September/2010
Edit Email Body	1	14th/September/2010
Edit Recipient's Address	0	9th/September/2010

Table 7.1 Test Result

The aim of the test was to debug the program so that it would meet all requirements specified under Core Features. Problems were corrected when identified.

7.2 Interface Testing

The user interface was tested on Mozilla Thunderbird. These tests included viewing the XUL interface and verifying all components that displayed on different tabs. Besides, some events such as clicked event were also tested in this step.

The program was installed on Thunderbird so that all components and files were tested. In this phase, every button and list needed to be tried.

7.3 Heuristic Evaluation

The heuristic evaluation is effective and inexpensive evaluation of user Interface design which was proposed by Jakob Nielsen. The system was evaluated by using ten principles [11] (these bullet points below were quoted from his book: *Usability Inspection Methods*, written by Jakob Nielsen, 2005):

- Visibility of system status
- Match between system and the real world
- User control and freedom
- Consistency and standards
- Error prevention
- Recognition rather than recall
- Flexibility and efficiency of use
- Help users recognize, diagnose, and recover from errors
- Help and documentation
- Aesthetic and minimalist design

The aim of heuristic evaluation was to locate the usability problems in user interface design and correct them.

7.4 Revision Control (Version Control)

Revision Control, also known as Version Control, is widely used in software development and management. It provides a solution to track the changes in a program's development. Also, the changes are stored in an orderly rather than chaotic structure. So, this enables the system to be recovered to a previous version.

More than 60 versions had been created during the project's development. The sequence of versions can be represented as a tree:

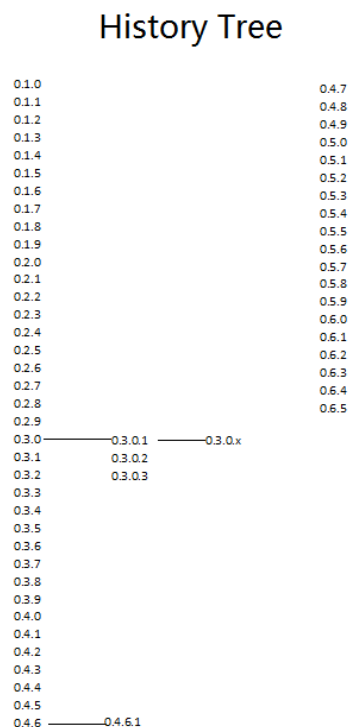


Fig. 7.2 History Tree

8. Discussion

This part will give the conclusion of this project, and present the issues. The possible future works will be expected at last.

8.1 Conclusions

One of the most significant conclusions in this project is that it provides a template for the other developers. With tidy and clear structure, the code can be easily understood. Besides, this add-on application is also a dedication to the researchers who are collecting email statistics from a number of subjects.

8.2 Issues

8.2.1 Code implementation Issue

There is trend for Mozilla developers to use Java as the main programming language. Using java language can benefit the artefacts in many ways since there are many limitations and constrains in JavaScript. For instance, the concepts in Java and JavaScript (JSON) are different. In JavaScript, one class can only have one class object. In a sense, it is more like a pointer which points to the class itself.

Furthermore, containers such as array list or hash map can be applied in Java while there are not available in JavaScript. However, array in JavaScript is not equal to the one in java because its length is resizable so that can be used as a stack.

Obviously, Java is more convenient than JavaScript in many ways. So, why not use Java in this project? The reason is that there are few materials about how to use Java on Mozilla's web site (MDC). The method of calling its interfaces is still not known. Rather, their publications use JavaScript as the only programming language.

More than 50 per cent latest add-on applications were developed using Java, that is to say, Java could take over the predominant position in few years. As to one of the future works, this add-on application could be re-programmed using Java.

8.2.2 Folder ID and Naming Issue

As mentioned in the previous Chapters, different email service providers vary in naming principles. For instance, some providers named 'outbox' as 'sent' or 'sent email'. This causes that the system requires the user to set the folder IDs manually. Is it possible for the system to automatically recognise the folder name? Yes, regular expression can be helpful in recognising folder names. However, applying regular expression is by no means easy due to limited methods in JavaScript. This is another reason for why many develop has abandoned JavaScript. Applying regular express will definitely cost plenty of time and increase at least 800 code lines. Thus it had to be left for future work.

8.3 Future Work

8.3.1 Using Java

Using Java can greatly minimise the workload and complexity of programming. Though most implementation details are still not known, it will hopefully become the main trend in Thunderbird add-on development.

8.3.2 Applying Regular Expression

If this application had been programmed using Java, the regular expression issue should have been solved. There is a big library of regular expression called “regex” in Java as well as in C++, Perl, Ruby, and Python.

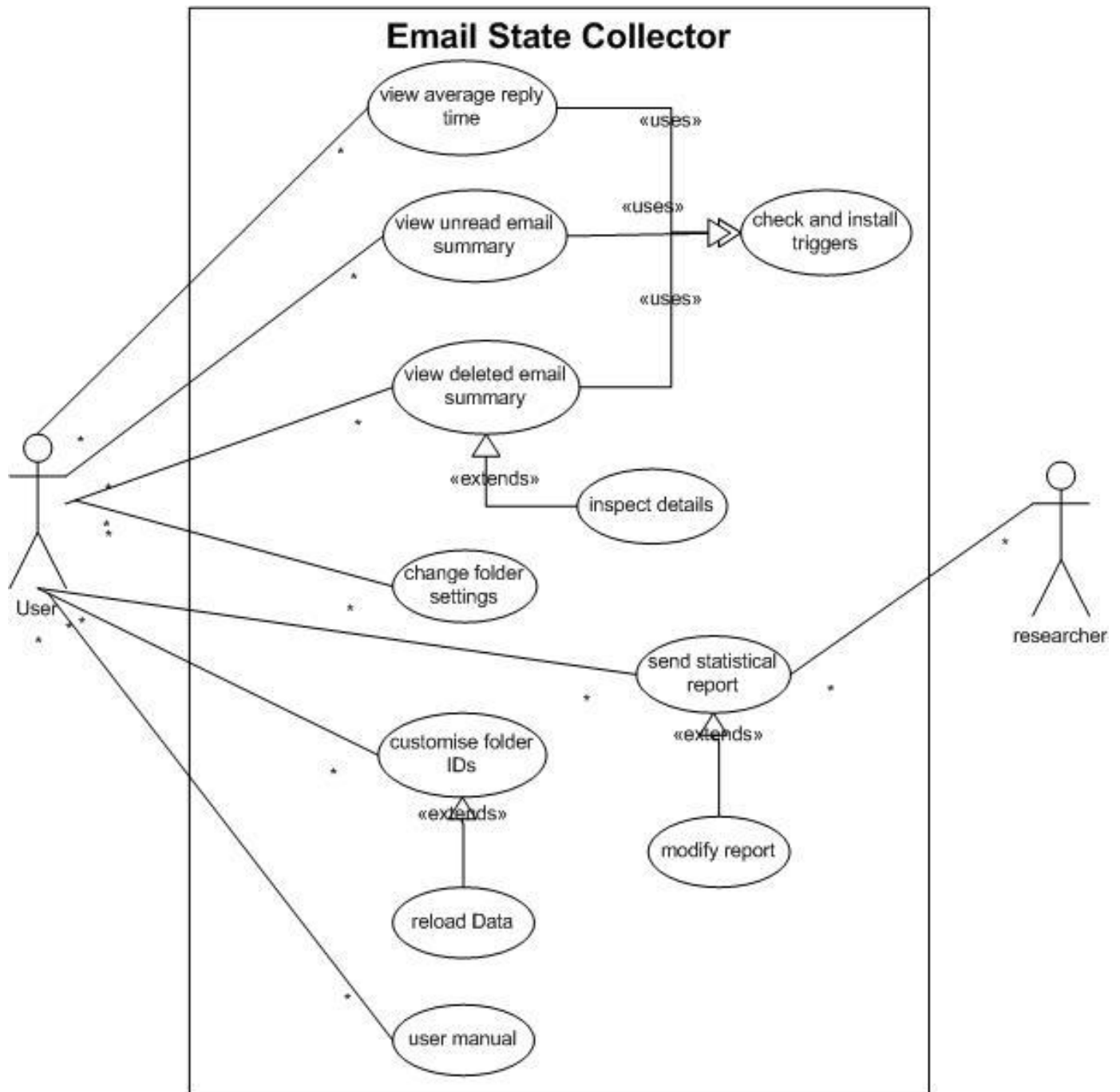
8.3.3 Survey

Due to the time limitation, launching a survey was impossible. First, the survey needs to be approved by the ethics committee of ANU. This could cost around 2-3 weeks since they review applications once a month.

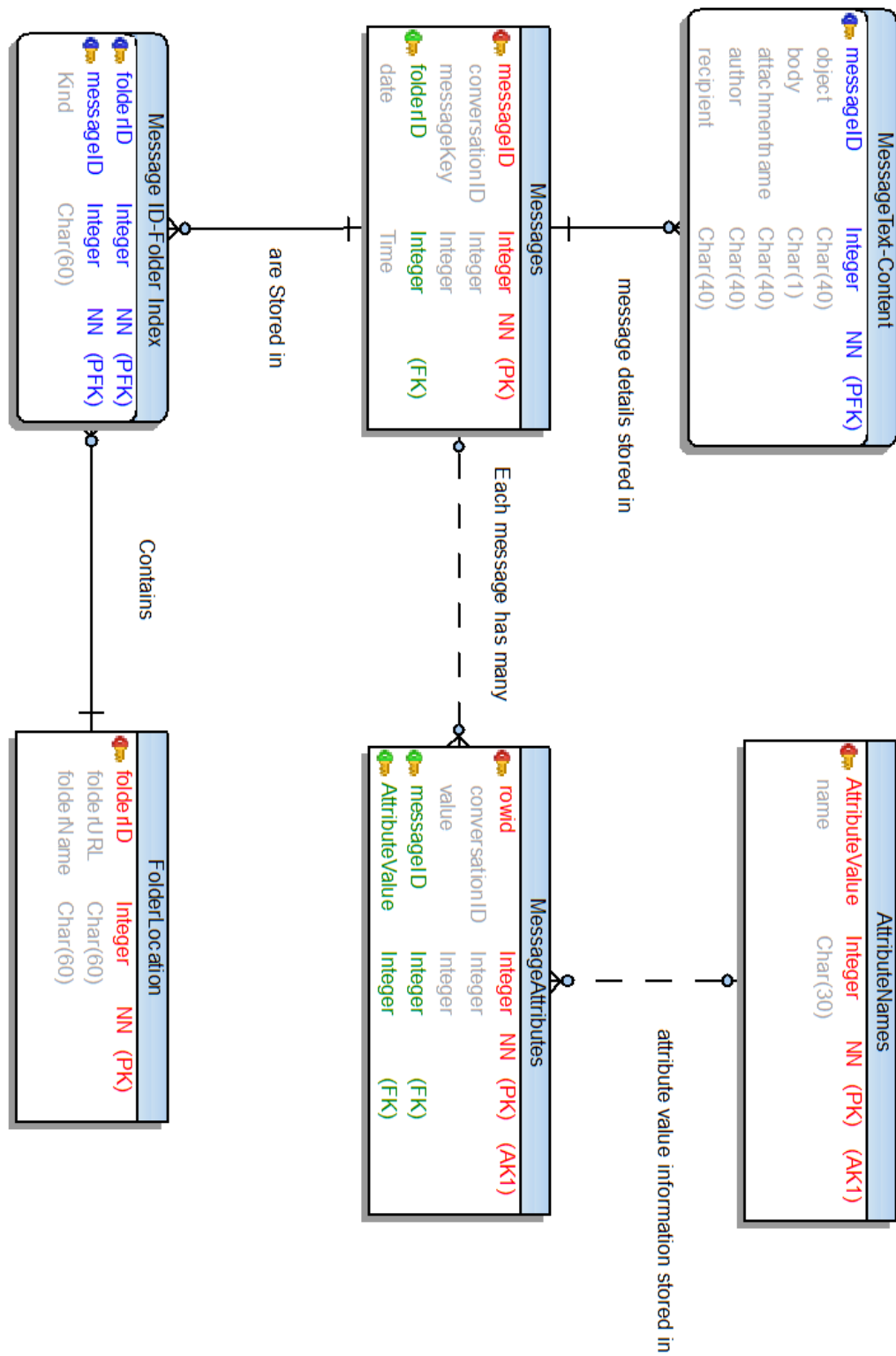
9. References

- [1]. *Understanding Email Use*; Written by **Laura A. Dabbish, Robert E. Kraut, Susan Fussell and Sara Kiesler**; Publish on *Conference on Human Factors in Computing Systems*, 2005.
- [2]. *Email and Webmail Statistics*; Written by **Mark Brownlow**; Published on *Email Marketing Reports*, 2009; URL :< <http://www.email-marketing-reports.com/metrics/email-statistics.htm> >.
- [3]. *An Evaluation of Identity-Sharing Behavior in Social Network Communities*, written by **Frederic Stutzman**, The University of North Carolina at Chapel Hill, 2005.
- [4]. *Facebook Plans to Make Money by Selling Your Data*, written by **Lidija Davis**, published on *www.readwriteweb.com*, 2009.
- [5]. *Used Electronic Mail to Conduct Survey Research*, written by **Liz Thach**, Published on *Educational Technology*, March-April 1995.
- [6]. *Professional JavaScript for Web Developers* (second edition); Written by **Nicholas C. Zakas**; Published by *Wrox Corp.* 2007.
- [7]. *Programming Firefox: Building Rich Internet Applications with XUL*; Written by **Kenneth C. Feldt**; Published by *O'Reilly*; April, 2007.
- [8]. *Learning XML*; Written by **Erik T. Rav**; Published by *O'Reilly*; September, 2003.
- [9]. *The Definitive Guide to SQLite*; Written by **Mike Owens**; Published by *Apress*; 2006.
- [10]. *Design of Everyday Things*; Written by **Donald A. Norman**; Published by *Basic Books*; 2002.
- [11]. *Usability Inspection Methods*; Written by **Jakob Nielsen & Robert L. Mack**; Published by *John Wiley & Sons, Inc.* 1994.
- [12]. *XUL Tutorial*: Written by **Neil Deakin**, Published by *MDC*; 2008.

Appendix 1: Full Size Figures



UML User Case Diagram



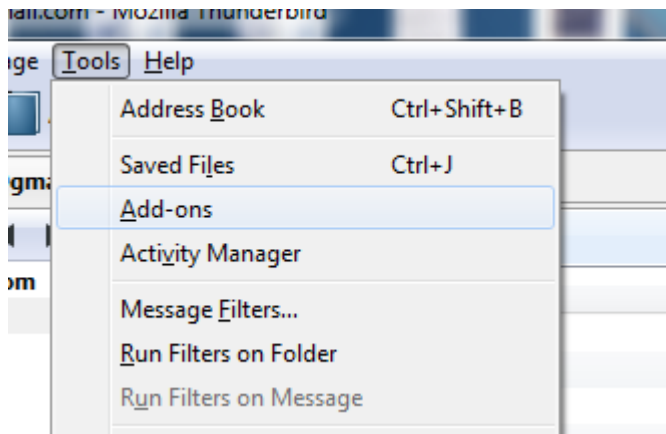
Database Structure

Appendix 2: User Manual

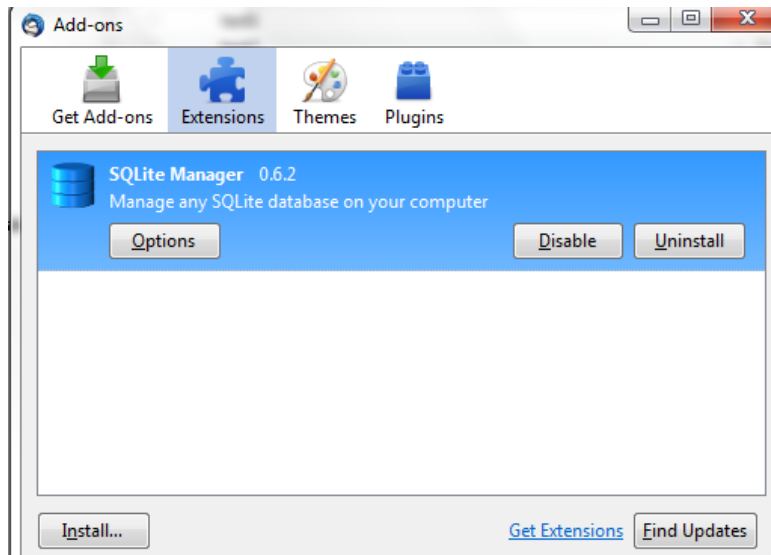
Install & Uninstall Application

Install

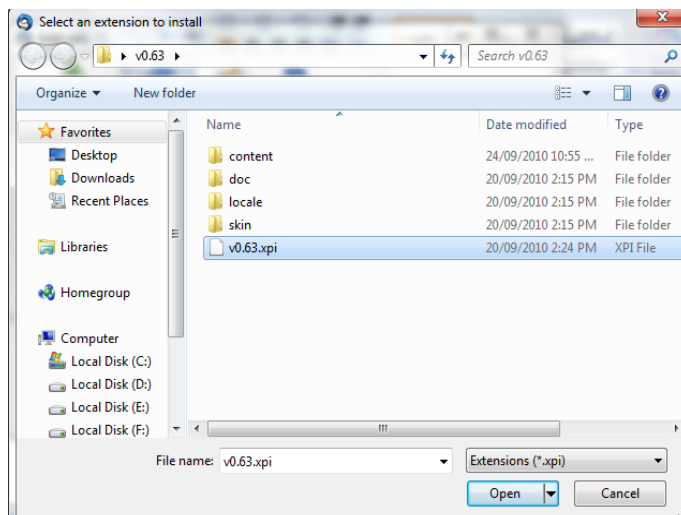
Start Thunderbird, and click Tools on menu bar, then click Add-ons.



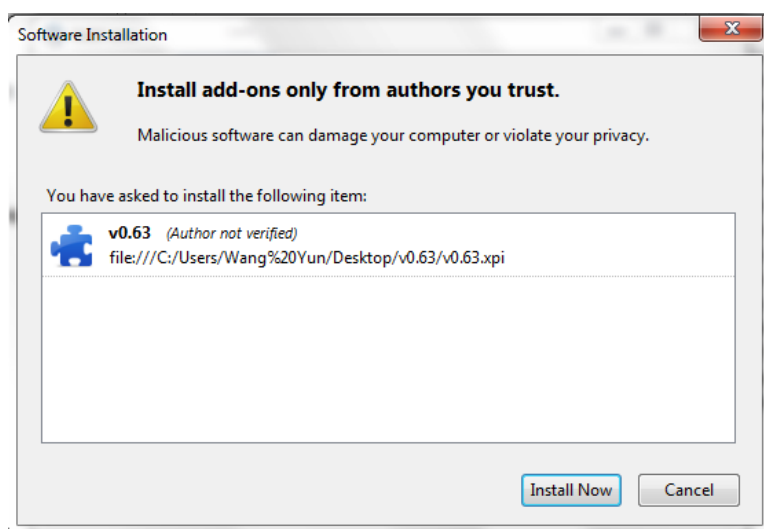
The system will prompt up a new window with four tabs. Click the Extensions tab, and then click on Install at the lower left corner.



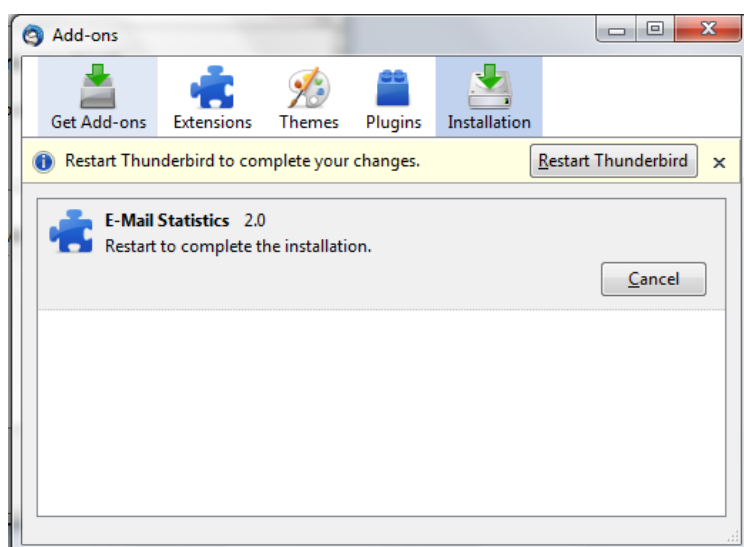
The system will proceed to prompt up a file-browsing window. Find the file you want to install and click open.



Then, click button install now.

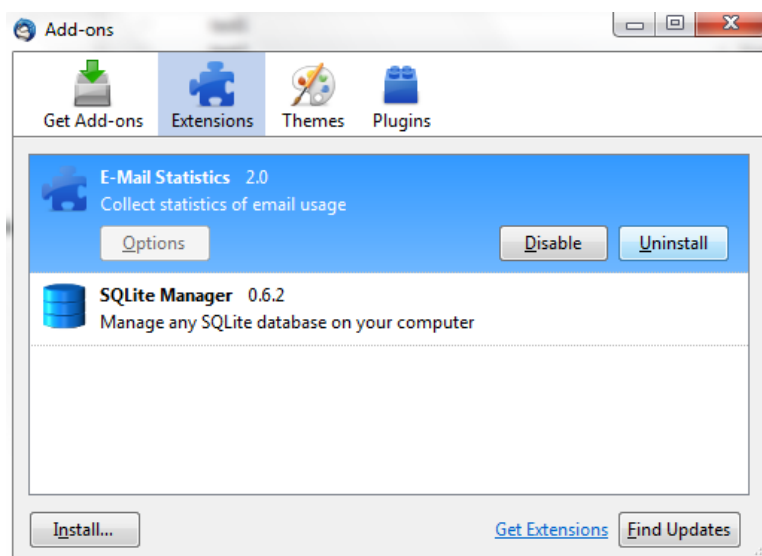


Finally, restart Thunderbird to finalise the intallation.

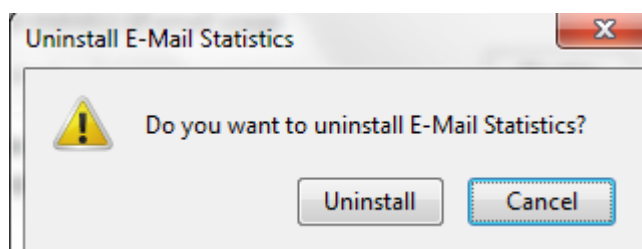


Uninstall

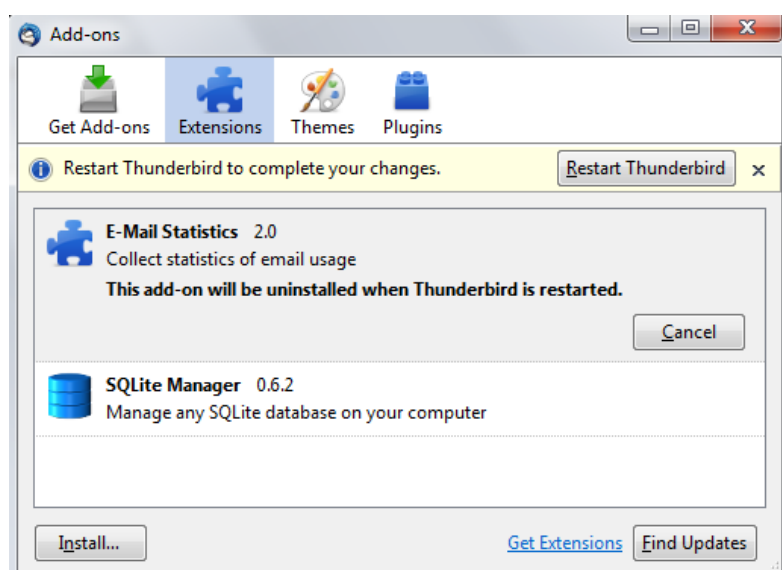
Similar to installation, the first step is to click "Tools" on the menu bar, then click "Add-on"



Click on the item you want to remove, and choose uninstall. Then confirm your action.

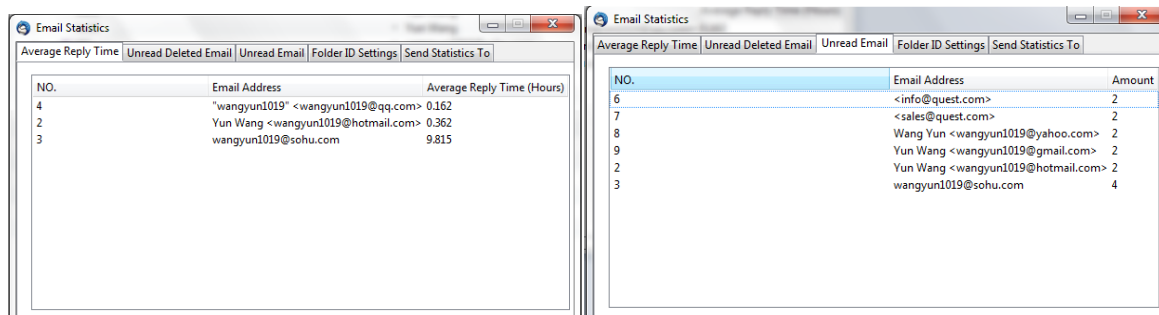


Finally, restart thunderbird.



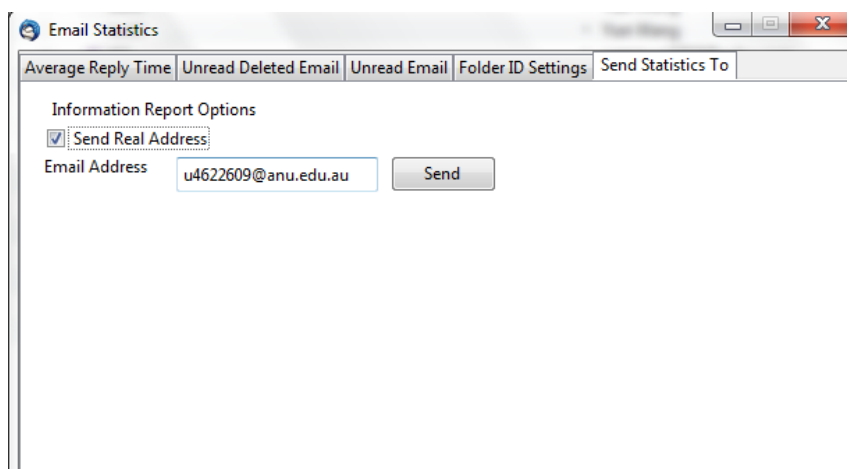
Viewing Statistics

By clicking on different tabs, you can switch between different kinds of statistics. You do not need to perform any other actions.

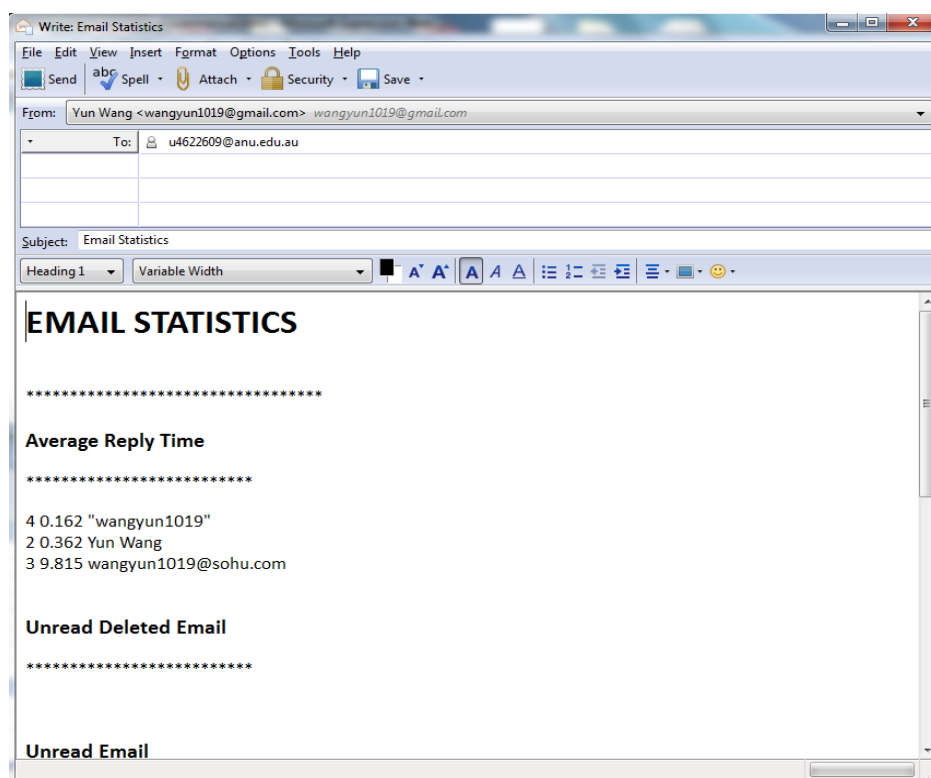


Customise and Send Report

Click on the last tab named "Send Statistics To". The recipient's address can be changed by modifying the string in the textbox.

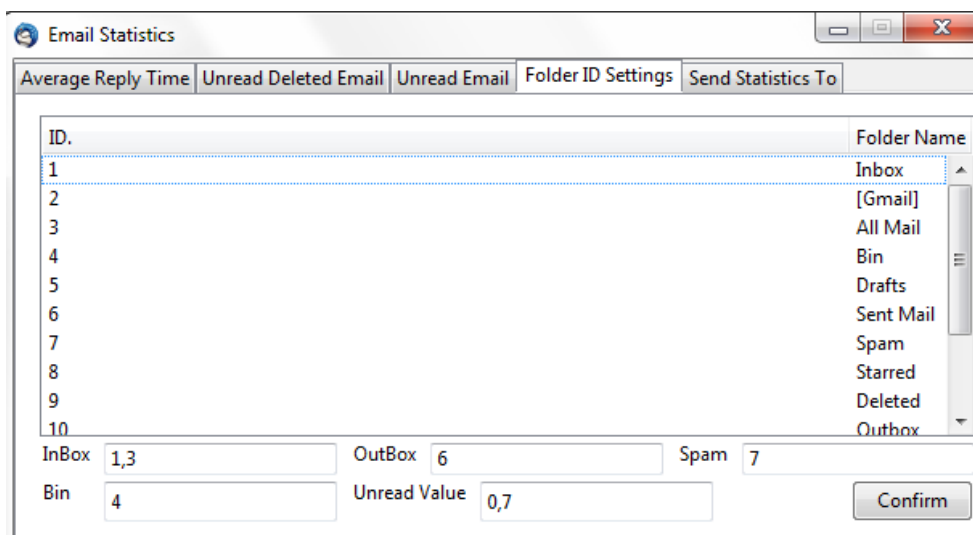


Click on the send button, the email body will be shown. You can also change the body to whatever you want



Folder ID Settings

Click on "Folder ID Settings", the dialogue will be shown.



This must be the most difficult part in this application since different email service providers vary in naming principles.

For instance, some providers name outbox as sent mail, sent email, sent or sent item. So, you need to group the Folder names which share the same meaning and put their IDs into the corresponding textbox.

Finally, click “confirm” button to rebuild the lists.