SET09102 Software Engineering

Jerzy Bajer 40428545

**Table of Contents**

[Introduction 3](#__RefHeading___Toc1089_975332164)

[1. Requirement specification 4](#__RefHeading___Toc1091_975332164)

[2. Class diagram 6](#__RefHeading___Toc1093_975332164)

[3. Testing 9](#__RefHeading___Toc1095_975332164)

[4. Version control 16](#__RefHeading___Toc1097_975332164)

[5. Evolution strategy 17](#__RefHeading___Toc1099_975332164)

[About 19](#__RefHeading___Toc1664_15503850)

# Introduction

Client Napier Bank requires service in order to allow validation, sanitization and categorisation of incoming messages in form of SMS text messages, emails and Tweets. Service called Napier Bank Message (NBM) must be created by 27.11.2020 15:00.

Service also called application will be created by one person only. The person creating this service will have to analyse, plan, design, develop and test the service. The person creating this service will have to full fill many different roles: project manager, analyser, designer, developer and tester.

The fact that this service will be created by one person only has the following consequences. Communication will be limited to contacting client in order to clarify ambiguities. Creator of this service will have full knowledge of analysis, planning, design, development and testing of this service. There will be no cooperation between team members as there will be no team.

Because of limited amount of time and resources this service will be created in a simple way. Everything will be kept as simple as possible. This will allow to avoid majority of errors and bugs. The focus will be to make this application as functional as possible, easy to use, flexible, fast with good look and to comply with all requirements. Project will be created with usage of Agile method. New features will be added at each iteration. Simple design and using Agile method will allow easier modification of this application in the future.

This document presents:

1. Requirement specification in form of user case diagram and additionally with full description of each requirement.

2. Class diagram presenting all classes, attributes and methods.

3. Testing describing testing strategy, methods and test cases.

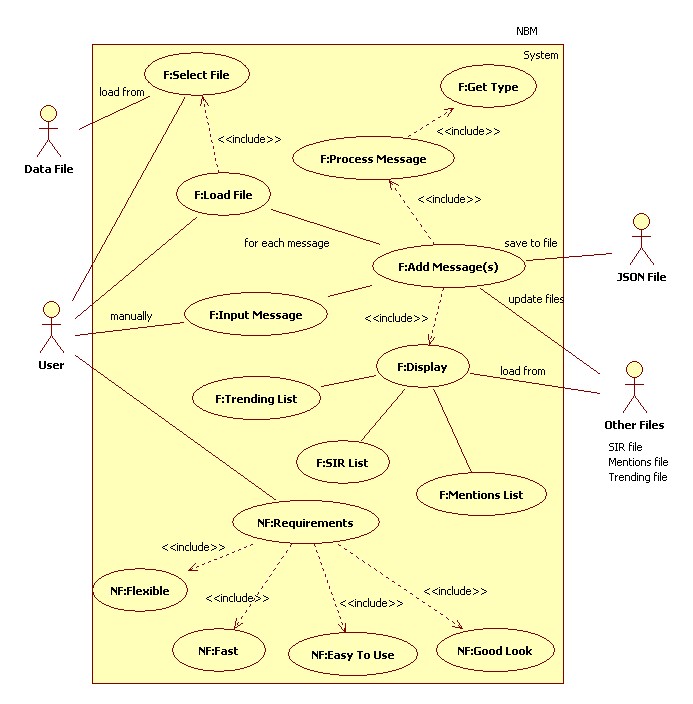
4. Version control used during creation of this service.

5. Evolution strategy.

About explaining flexibility of the program and how to use it.

# 1. Requirement specification

User Case diagram shown below (Figure 1.) displays functional and non-functional requirements.

Figure 1. Case Diagram

F: functional requirement

NF: non-functional requirement

**Constraints:**

- System must be finished by Friday 27 November 2020 15:00

- System must use JSON objects to store messages in file

**Functional requirements:**

- Program must have capability to accept messages in a form of Email, SMS, Tweet. Messages will be input in the form of the Message Header (text box) and Message Body (text box) and redisplayed in appropriate text boxes

- Program must have capability to accept data from file

- Program must have capability to detect type of message based on Message ID (“S”, “E”, “T” followed by 9 numeric characters)

- Program must have capability to process SMS properly – Sender (phone number), Message Text (maximum 140 characters) which may contain textspeak abbreviations. Textspeak abbreviations must be expanded to their full form enclosed in “<>”.

- Program must have capability to process Emails properly – Sender (full name and email address), Subject (maximum 20 characters), Message Text (maximum 1028 characters) which may contain URLs. URLs must be removed and replaced with “<URL Quarantined>”. Email can be divided into Standard and Significant Incident Reports. Significant Incident Reports will have Subject (“SIR dd/mm/yy”) and Message Body starting with sort code and nature of incident. For SIR emails sort code and nature of incident must be added to SIR list.

- Program must have capability to process Tweet properly – Sender (Twitter ID: @ followed by maximum 15 characters), Tweet text (maximum 140 characters.) which may contain Textspeak, Hashtags and Twitter IDs. Textspeak abbreviations must be expanded to their full form enclosed in “<>”. Hashtags must be added to a hashtag list that will count the number of uses of each to produce a trending list. Twitter IDs must be added to Mentions list

- Program must have User Interface (UI)

- Program must be able to display trending list, Mentions list and SIR list

**Non-functional requirements:**

- Program must look good visually.

- Program must be easy to use

- Program must be flexible

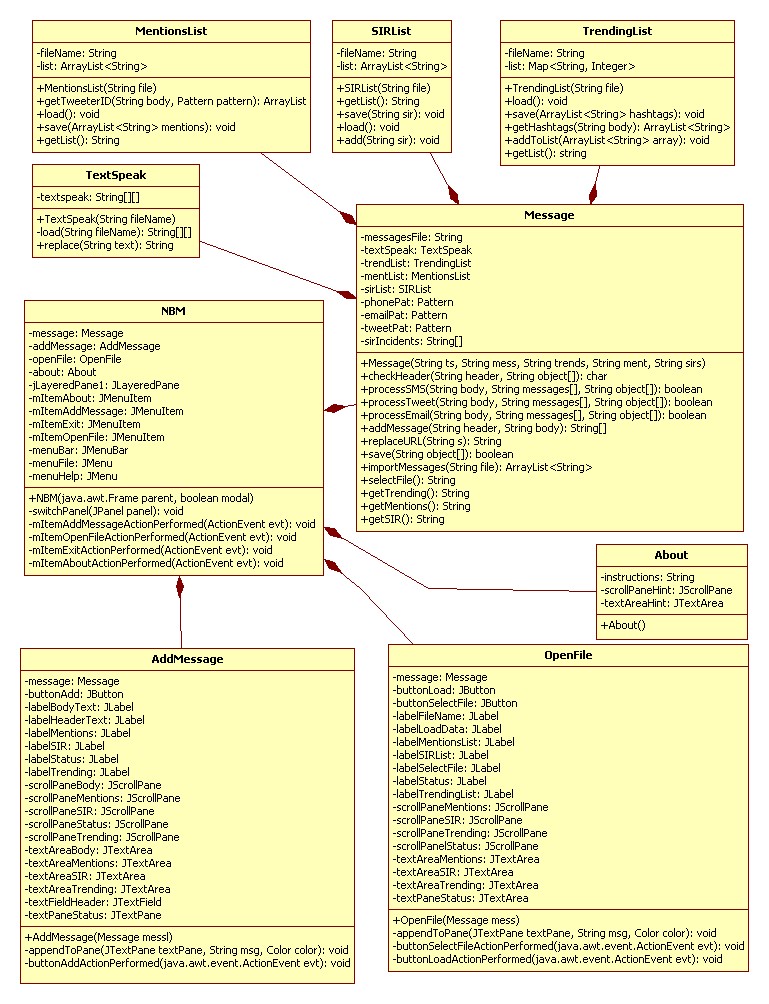
- Program must work fast

**Constraints:**

- Program must be finished by Friday 27 November 2020 15:00

- Program must use JSON objects to store messages in file

# 2. Class diagram

Figure 2. Class Diagram

As it can be seen in Figure 2. application NBM consist of nine classes.

TextSpeak class is responsible for replacing textspeaks with full explanations. File textwords.csv provided by client will be used to load all textspeaks and their explanations. This class is used in Message class when processing message.

MentionsList class is responsible for loading from file, saving to file, finding and returning Tweeter's ids in tweet message and returning Mentions list as a string. Mentions will be kept in file called mentions.txt. This will allow to load all mentions after user finish adding message(s). This class is used in Message class when processing message.

SIRList class is responsible for loading from file, saving to file and dealing with SIR emails (sort code and nature of incidents.) It also allows to return SIR list as a string. SIR will be kept in file called sir.txt. This will allow to display all SIRs after user finish adding message(s). This class is used in Message class when processing message.

TrendingList class is responsible for loading from file, saving to file and dealing with hashtags. It also allows to return trending list as a string. Trending list will be kept in file called trending.txt. This will allow to display trending list after user finish adding message(s). This class is used in Message class when processing message.

Message class is used to prepare and process each message properly. First this class allows to detect type of message and then it allows to process message properly. This class is also responsible for saving messages to file and loading messages from file. It also uses mentioned above MentionsList object, SIRList object and TrendingList object to update and display Mentions list, SIR list and Trending list.

NBM is main class which contains objects of other classes. This class is responsible for running whole application. When user starts program this class creates window and initializes all objects. This class is responsible for switching between panels. This allows to separate different panels / windows from each other. This class as it can be seen is relatively small.

AddMessage class is used to allow user to add single message. This class uses Message object to prepare and process message and then to display mentions list, trending list and SIR list.

OpenFile class is used to load messages from file. This class same as AddMessage uses Message object to select file to load data and then to prepare and process messages and then to display mentions list, trending list and SIR list.

About is class providing hints to user how to use this application.

As mentioned before this application will be created in a simple way. In summary there is one main class NBM which is responsible for running program. There are tree graphical classes responsible for displaying three different panels / windows. There is one class Message responsible for processing all messages and other task specified in requirements. Other classes are supporting Message class during processing each message.

Creating application in such way allows further modifications without any significant problems. Classes are generally small. Code is easy to be understood. Any further modifications should not cause many problems.

# 3. Testing

In order to make sure Napier Bank Message service works properly there will be used behavioural testing strategy. This strategy focuses on behaviour of the system. It pays attention to how system acts. Behavioural testing strategy focuses on tests from end user perspective.

Mainly there will be used defect testing focusing on detecting errors and bugs.

The objective of testing will be to design test cases in such way to find all potential errors and bugs and to make sure that application complies with specified requirements. Test cases will be created during requirement analysis.

Testing will start with unit testing. First each unit, each method will be tested by developer to make sure it works the way it should. This kind of testing will allow developer to find all bugs in logic.

Then developer will use integration testing to find potential bugs and errors in communication between units and modules.

Then tester will conduct system testing to make sure that this application meets all specified requirements and to make sure performance is satisfactory.

Finally tester will conduct acceptance testing in order to make sure that application meets customer requirements.

Developer will conduct unit testing with usage of white box method. White box testing is method in which tester has knowledge of the software. Tester can see code. Then developer will conduct integration testing with usage of black box testing method. Black box testing is method in which tester does not have knowledge of the software. Tester cannot see the code.

Additionally developer will write some automated test methods in JUnit. Those methods will be used to test some of the methods from Message class to find bugs in logic and to make sure that methods allow to accomplish their goals.

Tester will conduct system testing and acceptance testing with usage of black box method. Tester will conduct testing from perspective of end user trying to accomplish all common tasks end user may wish to accomplish.

Test cases:

Test case 1: Check results when user clicks File and then clicks Add Message

Test case 2: Check results when user clicks File and then clicks Open File

Test case 3: Check results when user clicks Help and then clicks About

Test case 4: Check results when user clicks File and then clicks Exit

Test case 5: Check results when user tries to add message with empty header and body

Test case 6: Check results when user tries to add message with empty header

Test case 7: Check results when user tries to add message with empty body

Test case 8: Check results when user tries to add message with incorrect message id in header

Test case 9: Check results when user tries to add SMS message with incorrect phone number

Test case 10: Check results when user tries to add SMS without message

Test case 11: Check results when user tries to add SMS with correct message id, phone number and message

Test case 12: Check results when user tries to add SMS with correct message id, phone number and message more than 140 characters

Test case 13: Check results when user tries to add Tweet message with incorrect Tweeter id

Test case 14: Check results when user tries to add Tweet without message

Test case 15: Check results when user tries to add Tweet message with correct Tweeter id and message

Test case 16: Check results when user tries to add Tweet message with correct Tweeter id and too big message

Test case 17: Check results when user tries to add Email message with incorrect email address

Test case 18: Check results when user tries to add Email message without subject

Test case 19: Check results when user tries to add Email message without message

Test case 20: Check results when user tries to add normal Email with everything correct

Test case 21: Check results when user tries to select file (File/Open File).

Test case 22: Check results when user tries to load messages (File/Open File).

Test case 23: Check results when user tries to add SIR email

Results of test case tests:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Date / Tester | Item to Test | Test Purpose | Expected Results | Actual Results | Comments |
| Test case 1 14/11/2020 Jerzy Bajer | Menu Item File/Add Message panel | Checking if program will switch to AddMessage panel | Should switch to AddMessage panel |  | Pass |
| Test case 2 14/11/2020 Jerzy Bajer | Menu Item File/Open File panel | Checking if program will switch to OpenFile panel | Should switch to OpenFile panel |  | Pass |
| Test case 3 14/11/2020 Jerzy Bajer | Menu Item Help/About panel | Checking if program will switch to About panel | Should switch to About panel |  | Pass |
| Test case 4 14/11/2020 Jerzy Bajer | Menu Item File/Exit | Checking if program exit | Program should exit |  | Pass |
| Test case 5 14/11/2020 Jerzy Bajer | Adding message with empty header and body | Checking if program will show error | Should show error header and body must not be empty |  | Pass |
| Test case 6 14/11/2020 Jerzy Bajer | Adding message with empty header | Checking if program will show error | Should show error header and body must not be empty |  | Pass |
| Test case 7 14/11/2020 Jerzy Bajer | Adding message empty body | Checking if program will show error | Should show error header and body must not be empty |  | Pass |
| Test case 8 14/11/2020 Jerzy Bajer | Adding message header incorrect message id | Checking if program will show error | Should show error that message id in header is incorrect |  | Pass |
| Test case 9 14/11/2020 Jerzy Bajer | Adding message incorrect SMS phone number | Checking if program will show error | Should show error that phone number is incorrect |  | Pass |
| Test case 10 14/11/2020 Jerzy Bajer | Adding message SMS no message | Checking if program will show error | Should show error that message is required in SMS |  | Pass |
| Test case 11 14/11/2020 Jerzy Bajer | Adding message correct all | Checking in program will display all OK | Should display message saved |  | Pass |
| Test case 12 14/11/2020 Jerzy Bajer | Adding message SMS with too big message | Checking if program will show error | Should display error to long message |  | Pass |
| Test case 13 14/11/2020 Jerzy Bajer | Adding message incorrect Tweeter id | Checking if program will show error | Should display error incorrect Tweeter id |  | Pass |
| Test case 14 14/11/2020 Jerzy Bajer | Adding message Tweet without message | Checking if program will show error | Should display error Tweet must have message |  | Pass |
| Test case 15 14/11/2020 Jerzy Bajer | Adding message Tweet with correct Tweeter id and message | Checking if program will show status is OK | Should display that Tweet was saved successfully |  | Pass |
| Test case 16 14/11/2020 Jerzy Bajer | Adding message Tweet with too big message | Checking if program will show error | Should display error too big message |  | Pass |
| Test case 17 14/11/2020 Jerzy Bajer | Adding message Email with incorrect email address | Checking if program will show error | Should display error incorrect email address |  | Pass |
| Test case 18 14/11/2020 Jerzy Bajer | Adding message Email with no subject | Checking if program will show error | Should display error subject is required |  | Pass |
| Test case 19 14/11/2020 Jerzy Bajer | Adding message Email with no message | Checking if program will show error | Should display error message is required |  | Pass |
| Test case 20 14/11/2020 Jerzy Bajer | Adding message Email with everything OK | Checking if program will show status OK | Should display message was successfully added |  | Pass |
| Test case 21 14/11/2020 Jerzy Bajer | Selecting file | Checking if program will allow to select file | Should select file and display full file path |  | Pass |
| Test case 22 14/11/2020 Jerzy Bajer | Loading messages | Checking if program will load messages from selected file and display status | Should load messages and display status |  | Pass |
| Test case 23 14/11/2020 Jerzy Bajer | Adding SIR email | Checking if program will add SIR email | Should display status is OK |  | Pass |

Test approach(s)

Above tests were conducted with usage of black box as mentioned above. NBM program was run in NetBeans and then certain tasks were performed in GUI. This allowed to check how the program will work in real life scenario.

Test Pass / Fail Criteria

* Program should react to each button being clicked by user, for instance once user clicks Add button program should add message.
* If user clicks exit button program should close
* If user clicks close button program should close
* Once user clicks Load program should load messages from selected file assuming user selected valid file first
* Program should catch all errors (empty header, empty body, incorrect phone number, incorrect Tweeter id, incorrect email address, empty subject, empty message etc.) and display error message stating what needs to be changed.
* Program should add message if all informations were inputted properly.

Test Deliverables

After conducting testing there will be delivered written report showing testing strategy and performed tests with results, pass or fail. There will be delivered working and tested application.

Test Environment / Staffing / Training Needs

All tests will be conducted by developer and tester Jerzy Bajer who will also create a testing report. Tester will use personal computer (PC) with Java IDE NetBeans 8.2 to perform all tests and Libre Office Writer in order to write testing report.

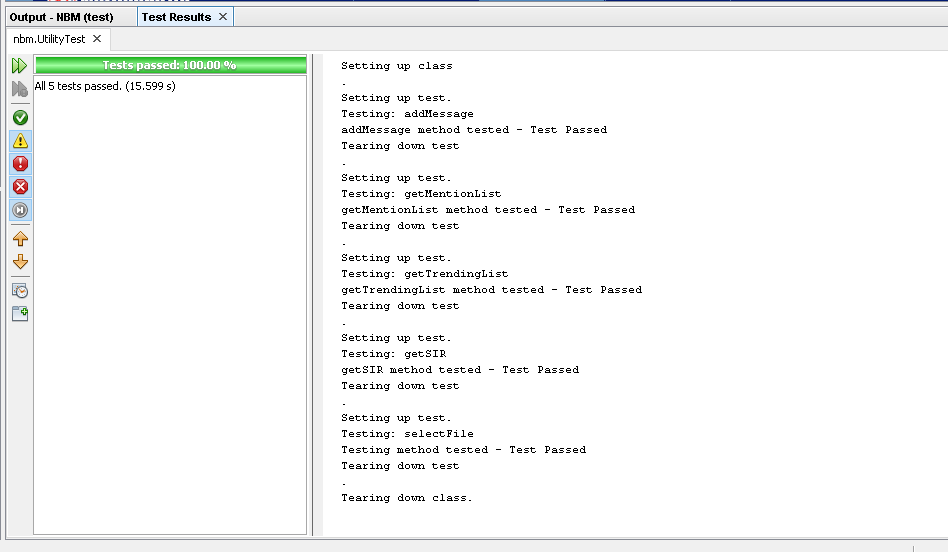
Figure 3. JUnit test – part 1

Figure 3 display results of running JUnit tests. As it can be seen all tests run successfully.

# 4. Version control

Version control is responsible for managing changes to software source code over time. Version control allows to keep track of every modification to the code. In case of mistake, developer can revert back to previous version while minimizing disruption to other developers in team. Developers working in teams modify code constantly by adding new features and changing existing ones. For instance one developer can make changes in one part of file while another developer may try to modify different part of the same file. Version control address those issues and allow to solve potential problems.

NetBeans 8.2 provide capability to use Git both locally and remotely. To use Git locally first local Repository must be created. This can be achieved by clicking Team Git Initialize Repository there will be created Git Repository in project folder. Once repository is created it is possible to Commit changes and to Revert to previous versions.

There are no doubts about benefits of using version control. However, as it was mentioned above this project will be created by one person only. There will be no cooperation between team members. There will be no team only one person completing the whole project from the beginning to the end. Additionally this project is relatively small which makes it much easier to control and complete. As it was mentioned above the whole system was designed in a simple way. So there should be no serious issues implementing the design. As a consequence there is no need to use version control while completing this project.

Lack of cooperation, relatively small project and simple design allows not to use sophisticated version control to keep track of all changes and to complete the project successfully. This is why basic version control will be used in this case.

At each iteration new features will be created. Those new features will be heavily tested. If new features pass all test they will be implemented to the system and whole system will be tested to make sure that new features were implemented properly and that all system works fine. In case some new features do not pass the tests they will be modified and tested again.

As it was shown in class diagram different parts of system will be divided into separate classes. This will allow much easier to maintain the whole system.

Keeping it simple is the key to successfully completing this project.

# **5. Evolution strategy**

Changes are inevitable. Everything changes, the same applies to applications. There are many reasons why NBM application may need to be modified.

Client may want application to meet new additional requirements. For instance client may want application to be capable to process different type of emails and to save data to server database.

Another reason may be that Napier Bank environment may change and client may need to modify application to follow the changes. For instance changes in the law may force client to adapt by modifying application.

It should not be the case however it is possible that some errors in application may need to be repaired. Testing should allow to find and eliminate all errors, but sometimes reality looks a bit different.

Another reason why client may need to modify application are changes in IT hardware and software infrastructure. For instance new computer’s hardware new operating system may require some modifications. Java generally allows huge flexibility but it is possible that at some point the NBM program may require some modifications caused by changes in IT hardware and software infrastructure.

Finally it is possible that client will need application to be capable to serve higher number of customers. Organizations rise and fall. If at some point Napier Bank grows the present application may not be any more satisfactory for client and may require some modifications.

NBM application was created in agile method which makes very easy to modify application in the future. In agile method at each iteration there are added new feature to application. Evolution is just a continuation of the development. Evolution in this case is just further development and improvement of application. This is why maintenance of this application should be relatively easy and not expensive. Additionally design of NBM application is simple and it is easy to understand code. Code is well commented. Sometimes it is better to spend more money during development in order to save later money during maintenance. It is like buying a car. New car or almost new car may be more expensive than old one, but it may allow to save huge amount of money on repairing it.

Of course it is hard to give precise cost of maintenance as it depends on many factors. For instance skills of staff responsible for maintenance, or application age may have huge impact on cost of maintenance.

Maintenance of NBM should focus on bug fixing, modification of application so it work in a new environment and adding new features in case of new requirements.

After some time if cost of maintenance increases significantly it may be good idea to use re-engineering. Re-engineering allows to reduce risk and cost. Parts of application which require change may be just rewritten. This may be much cheaper and less risky than creating new application from scratch.

# About

This application was designed to give some flexibility to the user:

**1. Adding single message**

Both Header and Body must be filled in to add message

**Header**

Header must contain valid Message ID. Message ID must start with single letter: ‘S’ or ‘s’ for SMS, ‘E’ or ‘e’ for Email, ‘T’ or ‘t’ for Tweet followed by nine numeric characters. Spaces ‘ ‘ and ‘-’ are allowed and are not counted.

For example:

S-000-000-001 = s 000 000 001 SMS

E 000-000 002 = e-000-000-002 Email

T-000-000-003 = t 000 000 003 Tweet

**Body**

* **SMS** message in body must start with valid international phone number followed my message.

International phone number may start with + or 00, spaces are not allowed, ‘-’ are allowed and are not counted. International phone number must have between 9 and 15 characters excluding ‘-’

For instance:

+44-131-11-22-11 = 0044131112211

Message must not be empty and must not exceed 140 characters. Textspeaks in message will be automatically replaced.

* **Standard Email** message in body must start with first-name last-name email subject in one line separated by space ‘ ‘ and email message starting from next line

Email must be valid email address

Message must not be empty and must not exceed 1028 characters. Any URLs contained in messages will be automatically replaced.

* **SIR Email** message in body must start with first-name last-name email subject typed in one line separated by space ‘ ‘.

Subject must contains SIR and date in following format dd/mm/yy. Both capital letters and lower case letters are allowed

For instance: SIR 22/11/20 = sir 22/11/20

Then in next line SIR email must have Sort code. Words sort code can be both capital case letters or lower case letters or can be omitted. Sort code must have the following format dd-dd-dd where d means digit

For instance: Sort code: 22-11-22 = 22-11-22

Then in next line SIR email must have Nature of Incident. Words Nature of Incident can be both capital case letters or lower case letters or can be omitted. Nature of incident can be both capital case letter or lower case letter Nature of incident can be one of the following: Theft, Staff Attack, ATM Theft, Raid, Customer Attack, Staff Abuse, Bomb Threat, Terrorism, Suspicious Incident, Intelligence, Cash Loss.

For instance: Nature of incident: Theft = theft

Then in next line SIR email must start email message. Message must not be empty and must not exceed 1028 characters (including Sort Code and Nature of Incident parts). Any URLs contained in messages will be automatically replaced.

* **Tweet** message must start with valid Tweeter ID followed by Tweet message.

Tweeter ID must not be empty and must not exceed 16 characters. Tweeter ID must start with @ followed by a maximum of 15 characters. Allowed characters include a-z, A-Z, 0-9 and \_

For instance: @john\_smith

Tweet message must not be empty and must not exceed 140 characters. Textspeaks in message will be automatically replaced.

**2. Loading messages**

Program allows to load messages from txt file. File must be prepared properly first.

Once user has prepared file to load messages user must select file in OpenFile window by clicking Select and choosing file.

Then user must click Load to import messages.