



UNIVERSITY *of* LIMERICK

O L L S C O I L L U I M N I G H

FINAL YEAR PROJECT REPORT

---

# A Command-Line Tool for Sending Webtexts

---

*Author:*

ROBERT O'DRISCOLL  
14150808

*Supervisor:*

DR. PATRICK HEALY

**B.Sc Computer Systems**

2018

## Table of Contents

<b>1</b>	<b>Project Summary</b>	<b>3</b>
<b>2</b>	<b>Introduction</b>	<b>3</b>
2.1	Motivation . . . . .	3
2.2	Overview of Existing Websites . . . . .	4
2.3	Problems with using the existing websites . . . . .	6
2.4	About this project . . . . .	7
2.5	Objectives . . . . .	7
2.6	About This Report . . . . .	8
<b>3</b>	<b>Research</b>	<b>9</b>
3.1	Researching Three . . . . .	9
3.2	Researching eir . . . . .	11
3.3	Researching CAPTCHAs . . . . .	14
<b>4</b>	<b>Design and Implementation</b>	<b>16</b>
4.1	Using HTTP requests . . . . .	16
4.2	Python . . . . .	17
4.3	Python Libraries . . . . .	17
4.4	Python Distribution . . . . .	19
4.5	Making the code extensible . . . . .	21
4.6	Sending Images . . . . .	22
4.7	Pasteboard . . . . .	24
4.8	Autocomplete and Comma-separated list of recipients . . . . .	25

4.9	Changes to Three's website . . . . .	26
<b>5</b>	<b>Analysis</b>	<b>28</b>
5.1	Comparison Between this Tool and Existing Websites . . . . .	28
<b>6</b>	<b>Conclusion</b>	<b>32</b>
6.1	Using Python . . . . .	32
6.2	Future Work . . . . .	33
<b>7</b>	<b>References</b>	<b>34</b>
<b>8</b>	<b>Bibliography</b>	<b>34</b>

## Acknowledgements

I would like to thank my supervisor, Paddy Healy, for proposing the idea for this project and also for his support throughout the project. He gave me the freedom to work on my own solutions but also offered valuable suggestions for how I might implement certain features.

# 1 Project Summary

The goal of this project is to create a command line tool that allows the user to send web-texts without using the website provided by the network e.g. Three, Vodafone etc. The purpose is to provide the user with a more streamlined experience through the automation of the web texting process. The user will not have to interact with the website directly, through a browser but instead, the tool will provide a more user-friendly interface that speeds up the process. A command-line tool of this nature also has applications in automation and could be integrated into larger programs.

The key motivation of the project is to create a viable alternative to the often tedious web interfaces created by the phone networks such as Three and Vodafone. Sending a single web text involves navigating multiple web pages including login screens and account dashboards. This project aims to create a tool that automates most of this process and sends a web text with at least user interaction as possible.

## 2 Introduction

### 2.1 Motivation

The motivation for this project is to develop a tool that allows users to send webtexts directly from the command-line. This tool would allow users to bypass the often tedious websites that are provided by mobile networks such as Three or eir. Ultimately, this tool should provide the user with a viable alternative method of sending a webtext that speeds up the overall process and offers a better user experience. The tool should allow the user to avoid much of the overhead involved in sending a webtext by automating tasks

such as logging in to the website.

## 2.2 Overview of Existing Websites

Webtexting is a useful service that is provided by mobile networks. Three, eir, Vodafone and others allow users to send regular SMS messages through their website, rather than from a mobile phone. Unlike SMS messages sent from a mobile phone, there is no charge associated with sending a webtext, although there is usually a limit to the number of webtexts that a user can send in a given period, e.g. 500 messages per month. Figures 1, 2, 3 and 4 are screenshots from Three's website and show the homepage, login screen, user dashboard and webtexting page respectively.

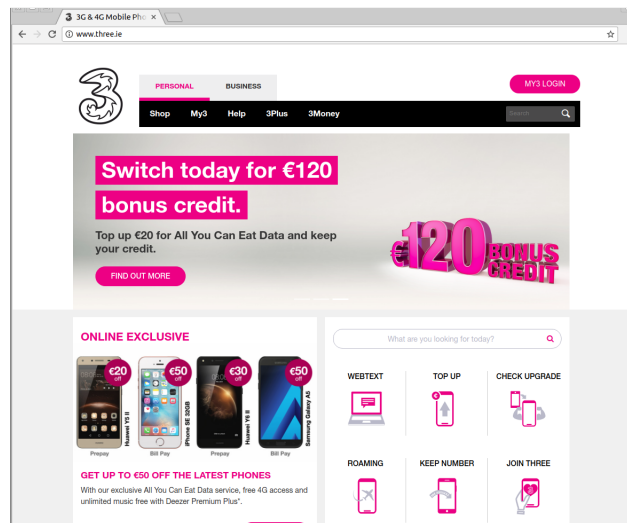


Figure 1: three.ie homepage

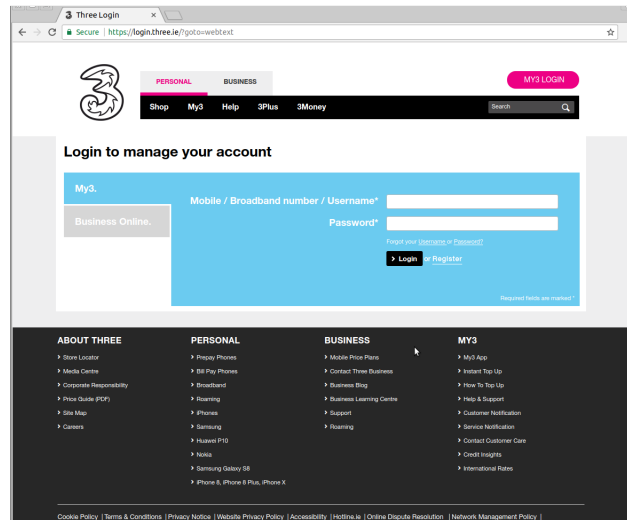


Figure 2: three.ie login screen

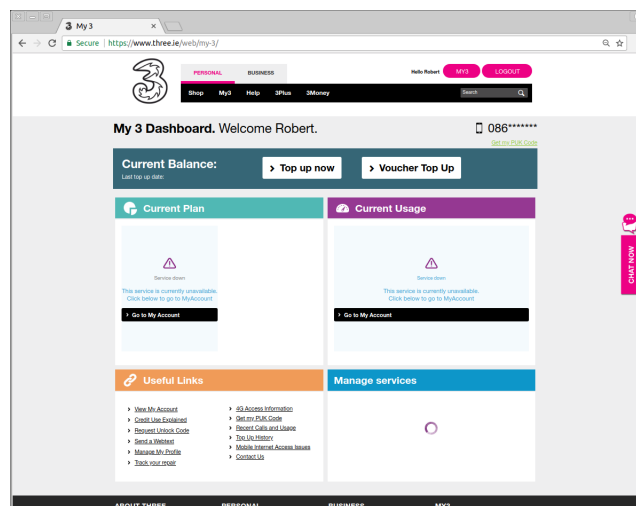


Figure 3: three.ie dashboard

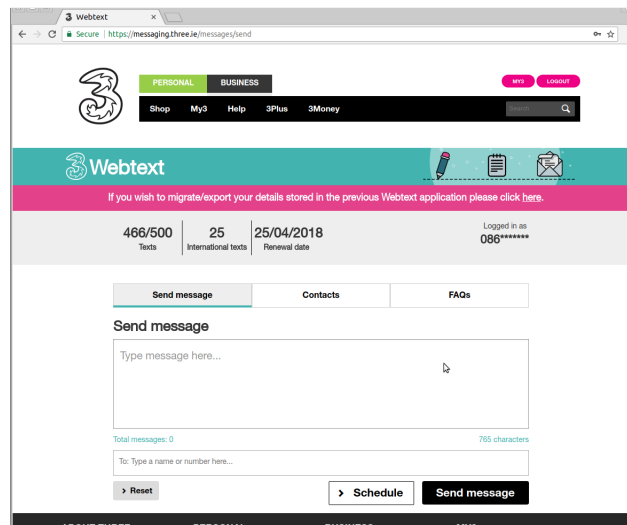


Figure 4: three.ie webtexting page

## 2.3 Problems with using the existing websites

Although webtexting is a useful service, sending a message using the network's website is often slow and tedious. In order to send a webtext, the user must navigate through series of login screens and dashboards, as shown in Figures 1 - 4. This process can also be difficult, especially for users who do not use the service often and who are unfamiliar with the process. The process of sending a webtext using the network's website can be off-putting and it is often easier and more convenient to send an SMS message from a mobile phone rather than use this website, even when the user has access to webtexting. This means that the user will be charged for the SMS message, even if it is a relatively small amount. This project aims to address some of the problems with these websites and make the process of sending a webtext as easy as sending an SMS from a mobile phone.



## 2.4 About this project

This project involves creating a tool that allows users to send webtexts from the command-line, without needing to interact with the network's website directly. This is accomplished by automating much of the webtexting process, for example the login process. The network's website is still used, but all interaction with this website is managed by the tool. This interaction is all done in the background. The user simply needs to supply information such as the message text and the telephone number to send the message to, and the tool manages the actual message sending. It is important to note that the networks have not exposed any API for sending webtexts, so this tool must implement functionality for interacting with the websites from scratch. Implementing this interaction, in an efficient and reliable manner is possibly the most important aspect of this project.

## 2.5 Objectives

The key objective of this project is to develop a tool that allows a user to send a webtext from the command-line, rather than needing to interact with the network's website directly. The tool must provide the user with a viable alternative to the existing websites. For example, if the tool was unreliable, a user would be forced to use the website whenever the tool was not working. Likewise, if it took much longer to send a message using the tool, a user would probably abandon it and choose to use the website instead.

Apart from simply sending a webtext, this tool should provide the user with additional features that are available from the networks' websites such as displaying the number of remaining texts that a user can send in a given time period. It should also implement certain checks to make sure that the login was successful. This is important because repeated incorrect login attempts may cause the user's account to be locked. There are also opportunities

to extend the functionality offered by the existing websites and offer more features to the user.

## 2.6 About This Report

**Research:** This chapter outlines the research that was done as part of this project. This includes researching the external and internal behaviour of Three's website, in order to understand how the tool developed for this project should function. It also includes researching eir's website from the context of extending the tool to work with multiple networks. Throughout the project, CAPTCHA challenges were a potential problem and still prevent the tool from supporting Vodafone webtexts. This section also presents some of the research that I did concerning CAPTCHAs during the project.

**Design and Implementation:** This chapter outlines the approach that was used when implementing various aspects of the project. It describes the technologies used and the justification for some of the design decisions that I made during the implementation. It also describes some of the features that have been added to the tool throughout the project.

**Analysis:** This section evaluates the success of the implementation of this project, particularly when compared to the existing websites provided by the Three and eir.

**Conclusion:** This section contains a brief reflection on this project. It describes my experience using `Python` and outlines some potential features that could be added in the future

## 3 Research

### 3.1 Researching Three

A large part of the research that needed to be done for this project involved becoming familiar with how Three's website worked, both from a user point of view and also technically. Understanding the way in which a user interacts with Three's webpage ( or any network's webpage) is very important. Not only is it necessary in order to create a working implementation, but understanding how the website is used leads to understanding what a user would want and expect from a tool that offers an alternative. For example, when using the website, a user must go to the login page and enter their username and password before they can send a webtext. This step can easily be avoided by automating the login process when using this tool. On the other hand, if the existing webpage was much easier to use than the tool implemented in this project, it would indicate that the tool had failed to meet the basic requirement of improving user experience. The websites offered by Three and other networks ultimately act as a benchmark to measure this tool against.

In order to create a working implementation of the tool, or to successfully update and maintain it, it is important to understand how Three's website works from a technical point of view. My implementation is based on interaction with the website through HTTP requests. The idea is to use Python to replicate the HTTP requests that are used normally when a user sends a webtext from the browser. In order to study these HTTP requests, I used the developer tools in the Firefox web browser. This tool allows you to monitor the requests that are being sent as you interact with a website. Most web browsers offer similar features but I found Firefox offered the most detail and was the most useful for this task.

This tool allows you to easily filter requests based on the resource that is being accessed. For example, it can be set to only show requests for **HTML** pages and ignore requests for resources such as **CSS** or **JavaScript** files as well as images. For the implementation of this tool, there is no reason to replicate these requests as resources such as images serve no purpose when the process is being automated. The tool should only replicate the **HTTP** requests that are absolutely necessary in order to successfully send a webtext. It is important to mention that the fact that this tool limits the number of requests to a minimum means that it is fundamentally more lightweight than the existing website.

Identifying the requests that are necessary is a very important step in the process of implementing or updating this tool. Requests for resources such as images and stylesheets can be excluded immediately. The remaining requests can then be examined further to determine whether or not they are needed. One way of judging these requests is by looking at the parameters that are passed. Discovering when the important details such as username, password and message text are sent can help identify what requests need to be replicated in the **Python** script.

Once the relevant **HTTP** requests have been identified, they must be successfully replicated. The Firefox developer tools allow you to clearly see the details of each request. This includes the URL, the headers and a list of the parameters that are passed with the request. It is also possible to edit and resend the requests in order to gain a greater understanding of how they work. For example, some parameters may change between requests or may not mean what you expect. Testing these requests directly from the browser is a very useful feature.

The Firefox developer tools can also be used when attempting to extract data from **HTML** pages. An important piece of information is the number of remaining webtexts that a user can send. **Python** can find this information

by parsing the `HTML` page that is included in a response from the website. In order to do this, you must determine what `HTML` element displays the user's balance. Firefox allows you to click on an element on a web page and finds that element in the source code for the page. This allows you to identify the element from `Python` using some identifying information such as the name of the class used in the `HTML` code.

### 3.2 Researching eir

In order for this tool to be more flexible, it must be able to be adapted/extended to work with other networks besides Three e.g. eir. I choose to implement functionality for eir as their website does not use CAPTCHAs and I was able to use another student's account in order to test the implementation.

Creating a working implementation of the tool for eir involved much of the same work as was needed for Three. It was a lot easier to implement for a second network as I had already implemented a lot of the features common to both networks and I was more familiar with the process involved. Once again, I used the developer tools in Firefox to study the `HTTP` requests that were used to send a webtext. I then used `Python` to replicate these requests, using the script I had developed for Three as a template.

Although the fundamental structure of the webtexting process was the same for both Three and eir, there were a few differences that I encountered. Eir uses a lot of `JSON` for both `HTTP` requests and responses. `JSON` (JavaScript Object Notation) is a way of structuring data in such a way that it can be easily converted to `JavaScript` Objects. For example, while Three uses simple query strings to pass parameters such as the message text and the recipient's phone number to the server, eir encodes this data as a `JSON` string. This does not have a major impact on the implementation for sending `HTTP`

requests as the code for sending parameters in these two formats is almost the same when using the request library:

### Using Query String:

```
requests.post(url,data={"key1":"value1","key2":"value2"})
```

### Using JSON:

```
requests.post(url,json={"key1":"value1","key2":"value2"})
```

The use of JSON makes more of a difference when dealing with responses. For example, this tool uses the data contained in the HTTP responses that are returned by the server to find information such as the remaining webtexts available for the current user. When a webtext is sent using Three's website, the server responds with an HTML page. This HTML page contains the information that we want to find i.e. the user's remaining webtexts. In order to extract this information, we must parse the HTML file using a library such as *BeautifulSoup*. Before any Python code is written, however, we must find some way to specify the exact HTML element that contains the information. The Firefox developer tools can be helpful for this step. We can then tell *BeautifulSoup* to find the element for us by describing it e.g. the first div box with a certain class name. Depending on how the element is defined in the HTML code, and how many other elements with similar attributes there are, this can be difficult and this implementation is very vulnerable as minor changes to the HTML page in question can cause *BeautifulSoup* to identify the wrong element or not find any element. When implementing this feature, eir's website is more suited to working with Python. Instead of returning an HTML page, which must then be parsed to find information, eir responds with raw data in JSON form. While the purpose of HTML is to define how content is displayed to the end user, JSON is meant to define data for use with pro-

programming languages. JSON is more Python-friendly and can be easily parsed to find the required data.

Another difference between eir and Three is that the HTTP request that is used to send a webtext using eir requires the user's phone number. The URL that is used in the request includes the number e.g.

`https://my.eir.ie/mobile/webtext/mobileNumbers/+353851234567/messages.`

In order to replicate this request, the Python script must, therefore, know the user's number. There are two main solutions to this.

The first is probably the more obvious, which is to simply require eir users to enter their phone number along with their username and password. This number could be stored in a configuration file so that they would only need to enter it, along with other login details on the first use and they would be read in automatically the next time the program was run. This solution may seem fine, however, it does present a problem. A large part of the program e.g. user interface, code to parse configuration files etc. is almost exactly the same whether Three or eir is the network being used. Therefore, it is ideal that this code can be written once and shared by the different networks. If the user's phone number was included in the information needed by the tool, it would mean that eir and Three could no longer share a common interface as eir would need an extra piece of data.

The second solution solves this problem. When a user logs into eir's website and sends a webtext, they do not need to enter their phone number. Instead, this information is retrieved from the server along with other account details such as their name, current plan etc. Once this request has been identified, it can be replicated in the Python script and used to retrieve the user's number so that it can be used later on. The data is returned in JSON format and can easily be parsed to find the phone number. It also contains other data

that could be used when implementing features in the future. This approach does mean adding an additional HTTP request to the procedure for sending a webtext using `air` but it means the code can be better designed and less information is required from the user.

### 3.3 Researching CAPTCHAs



Figure 5: example of a CAPTCHA challenge (Captcha.com, 2018). Screenshot by author.

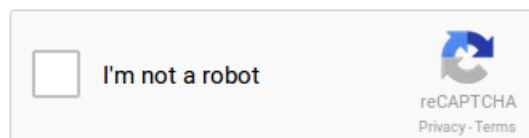


Figure 6: Google's reCAPTCHA (Google.com, 2018). Screenshot by author.

CAPTCHAs (Completely Automated Public Turing Test to Tell Computers and Humans Apart) are challenges embedded on a webpage that are used to determine whether a user is genuinely human or a bot. They are primarily used to prevent malicious activity such as spam or automatic account creation. This is done by requiring the user to complete a task that is easy for humans but difficult for a computer. A common example of this is requiring the user to read distorted text and type the same thing. This is relatively easy for most people to do, however, it would require advanced character recognition software in order for a computer to attempt the challenge.



CAPTCHAs are not only relevant to security but also to the field of Artificial Intelligence. It has been stated that "any program that passes the tests generated by a CAPTCHA can be used to solve a hard unsolved AI problem" (Von Ahn *et al.* 2004). The observation is that if a computer program was developed that could successfully solve the CAPTCHA challenges, it would also mean that progress had been made in artificial intelligence.

Some mobile networks e.g. Vodafone have added CAPTCHAs to their websites, preventing attempts to automate the webtexting process. Some tools similar to the one being implemented in this project have stopped working as a result of this. Fortunately, other networks such as Three and eir have not chosen to add CAPTCHAs to their websites. From reading forums and discussions online, it is clear that Three are aware that users want to be able to use third-party applications to send webtexts. Three moderators have responded to users who were experiencing problems with these third-party applications which suggests that they have no problem with people using them. The fact that they have chosen not to add CAPTCHAs suggests that they recognise that it would upset these users. It is therefore unlikely that they will choose to add CAPTCHAs in the near future.

One solution to solving CAPTCHAs that I considered was to download the image, display it to the user and have them enter the text from the command-line. However, implementing this proved troublesome as the CAPTCHA image is generated dynamically and changes each time the page is loaded. In addition to this, the CAPTCHA challenge is designed to detect non-human behaviour so things, like not typing on the web page or performing actions too quickly, would likely cause the test to fail. Newer implementations of this test such as Google's reCAPTCHA are more complicated and would not work with this approach.

## 4 Design and Implementation

### 4.1 Using HTTP requests

For the implementation of this tool, I chose to use HTTP requests as the primary mechanism for interacting with the network's webpage. Another potential approach was the use of browser automation. This involves mimicking the behaviour of a user interacting with a website through the browser. This includes mimicking actions such as button clicks, typing and pauses to replicate the speed at which a user would realistically be doing these actions. Browser automation has applications in testing, especially if you want to test the front end of the website e.g. check that HTML elements are responsive.

While this approach would potentially work for this project, there are a few major disadvantages. From a performance point of view, this approach would be very inefficient. There would be a large overhead involved in loading the resources on the page e.g. images, styles and also involved in controlling user actions e.g. button clicks. The code for this implementation would also be much longer and more difficult to maintain as it would require every action and every HTML element involved to be specified. Maintenance would be a big issue with this approach. While the network may change the website's backend, and the format of the HTTP requests involved, they are much more likely to change the frontend and likely to change it more frequently. A small change to any HTML element that is referred to in the code would break the tool. Therefore, while browser automation has applications in other areas, it would be a bad choice for this type of project. Ultimately, using HTTP requests directly makes a lot more sense and is probably the best approach to use.

## 4.2 Python

During the early stages of the project, it became clear that Python would be the best choice for this type of project. It has a wide variety of useful libraries available for the tasks involved in this project and its popularity means it is a valuable language to learn. There was also a choice to be made between Python 2 and Python 3. Ultimately, Python 3 was a better choice as it provides better support for the libraries used in the project. There is also no need for support of legacy features that would require Python 2.

## 4.3 Python Libraries

**Requests:** At the core of the implementation for this project is a Python library called Requests. This is an HTTP library that allows you to create GET, POST, PUT and DELETE requests from a Python script. It also allows you to manage cookies and sessions.

**Beautiful Soup:** *BeautifulSoup* is a Python library that allows you to parse HTML and XML documents. This can be used to extract data from a webpage.

**ConfigParser:** *ConfigParser* is a Python library that allows you to both create and parse configuration files in .ini format. This format stores configuration data in a file using a key, value method. Sets of key-value pairs can be split into different sections. This tool uses a .ini file to store a user's login details:

```
[three]
password = abc
username = user123@email.com
```

```
[eir]
```

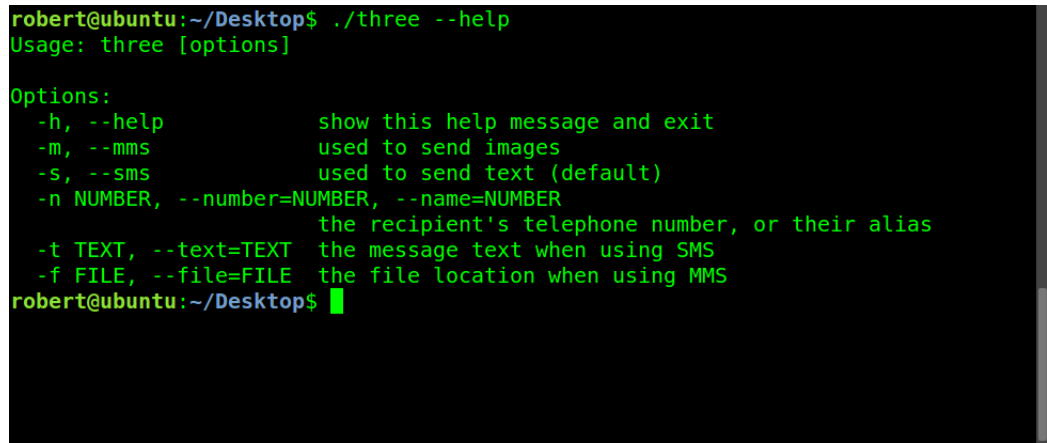
```
password = def
username = user456@email.com
```

Each network has its own section in the configuration file. This would allow a user who had accounts with more than one network to store different sets of login details simultaneously. The .ini file format is human readable, human editable and easy for a program to parse. It is also platform independent and is commonly used on both Windows and Linux systems. For example, PHP uses a php.ini file to store configuration data. This tool also uses the .ini configuration file to store an address book using a key-value pair of phone numbers and aliases.

```
[contacts]
john = 0871234567
bill = 0863456789
```

**optparse:** `optparse` is a `Python` library that helps you to manage command line arguments. It makes it easy to add flags for different options, for example, telling the script to use file sending rather than simple text. It can also generate help messages:

**readline:** `readline` is a `Python` module that that this tool uses to accomplish autocomplete when a user is entering the name of a contact. The script creates a list of possible names from the configuration file. The completion functions of the `readline` module are used to provide tab autocompletion of the user's input based on the list of names that was created.



```
robert@ubuntu:~/Desktop$ ./three --help
Usage: three [options]

Options:
  -h, --help            show this help message and exit
  -m, --mms             used to send images
  -s, --sms             used to send text (default)
  -n NUMBER, --number=NUMBER, --name=NUMBER
                        the recipient's telephone number, or their alias
  -t TEXT, --text=TEXT  the message text when using SMS
  -f FILE, --file=FILE  the file location when using MMS
robert@ubuntu:~/Desktop$
```

Figure 7: screenshot from tool showing help text

## 4.4 Python Distribution

Although **Python** is an interpreted language, there are occasions when it is useful to create an executable from the scripts. In many cases, the developers of a **Python** program may want to hide the source code from the user. If the program is distributed as a **Python** script, anyone can open the file and read or edit the source code. With an executable, the source code can be protected. Many executables created from **Python** scripts bundle all necessary libraries into a single executable file, making it easier to distribute as the user does not need to find additional libraries separately. The executable can run even if these libraries are not installed on the system. Some tools allow you to create an executable that does not need **Python** to be installed.

**Cx\_freeze:** *Cx\_freeze* is a cross-platform tool for freezing **Python** scripts into executables. It makes it easy to install the program on a system. Although I was able to use it to install my tool successfully on both Windows and Linux, I found that the executables which were created could not be run on a machine that did not have *Cx\_freeze* installed.

**Py2exe:** *Py2exe* is another tool for creating executables from **Python** scripts.

I was able to successfully create executables using this tool, however, it only supports Windows and not Linux. Although the tool developed for this project is designed to be cross-platform, it is targeted mostly at Linux systems. Therefore, *Py2exe* was too limited to use for this project.

**Pyinstaller:** *Pyinstaller* was the most successful tool for creating executables. It works on both Linux and Windows systems and the executables created can run on most machines, without requiring *Pyinstaller* to be installed. I was able to incorporate *Pyinstaller* into a bash script that can be used to easily build the executables from the **Python** code. I did encounter some problems running the executables created by *Pyinstaller*. When I attempted to run these executables on a machine running Fedora Linux, the program exited with an exception. I discovered that this was a known issue with *Pyinstaller* related to SSL certificates and the transition from **Python** 2 to **Python** 3. When I ran the **Python** script directly on the same machine, there were no issues.

**Build Script:** In order to make it more convenient to create executables for Three and eir, I created this small bash script. The script uses *Pyinstaller* to create executables from **Python** source code. It builds the executable for Three, and then changes the **Python** code temporarily to use eir instead. If support for another network was introduced, this step would be repeated until all executables were created.

```
#!/usr/bin/env bash
```

```
#build executable for three network
```

```
pyinstaller webtext_main.py --onefile -n Three --clean --distpath ../
```

```
#build executable for eir network
```

```
sed s/three/eir/ < webtext_main.py > webtext_main_temp.py
```

```
pyinstaller webtext_main_temp.py --onefile -n eir --clean --distpath ../
```

```
rm webtext_main_temp.py
```

## 4.5 Making the code extensible

An important aspect of this project is to develop a tool that can easily be extended to work with multiple networks. It is important to make the code as reusable as possible. To achieve this, the code is split into two major components. The first is the core functionality for sending a webtext. This is implemented separately for each network. The idea is that it should be possible to pass a username, password, message text and recipient's phone number to this component and have it send a webtext.

The second component in this tool is the code for the user interface and includes functionality for reading configuration files etc. This component contains the functionality that is shared by all networks. If support for a new network is added to this tool, no code in this component should need to be modified.

The core functionality of the tool is implemented using the Object-Oriented paradigm. In other words, an Object is defined for each network, that can be used by the user interface side of the implementation. The network that is used depends on the type of Object that is created. For example, to send a webtext using Three, a **Three** object is created.

In a programming language such as **C++** or **Java** that uses static typing, the classes that represent each network would be defined as subclasses of a network base class. They would be forced to implement certain methods that were common to all networks. The client code would create an object with the base class as its type. The subclass, and therefore the network, that was used could be chosen at runtime. Interfaces are used in **Java** in order

to assure client code that certain behaviour is available from a class that implements that interface. This allows the client code to use the interface at compile time and determine the concrete class at runtime. **Java** requires interfaces to be used because there is a certain amount of type checking that is done at compile time.

**Python** is a dynamically typed language. Interfaces are not necessary in order to defer the type of an object until runtime. Abstract Base classes have been added to **Python**, but, at least in the context of this project, there doesn't seem to be much of an advantage to using them.

If a developer was extending the code to add support for a new network, they would create a new class for that network. This class would be required to implement certain features that are used in the client code. If interfaces, or an equivalent were used to define a set of features that must be implemented by new network classes, and the developer forgot to add one of these features it would cause an error at run-time. In this scenario the error would state that the class was of network type but did not implement a certain method. If the developer was adding a new class and interfaces were not used, a missing feature would result in a run-time error which would state that a certain method had been called but didn't exist. These situations are effectively the same. Because **Python** is interpreted and dynamically typed, there is no clear advantage to explicitly defining an interface or abstract base class


## 4.6 Sending Images

In addition to sending text messages, mobile networks typically allow users to send other forms of messages, e.g. images, video. This is normally done using MMS (Multimedia Messaging Service) which is an extension of SMS (Short Message Service). This is a useful service, however, the messages sent using the webtext service provided by the networks are restricted to text



only. There is no way to send an image via the network's website. In order for the command-line tool to be able to send images, it needs to use more than the existing website provided by the network.

In this project, the ability to send images is achieved by uploading the image to a location online and sending a link to that location as a text message. The recipient can view the image that has been sent to them by clicking on the link.

A screenshot of a terminal window with a black background and green text. The prompt is 'robert@ubuntu:~/Desktop\$'. The command entered is './eir -m -f csis.png -n 086 [redacted]'. The output shows the tool logging in to 'eir.ie...', logging successfully, sending a webtext (1 of 1), and then displaying remaining text counts: 'Remaining texts: National: 255/300', 'International: 100/100', and 'Overall: 355/400'. The prompt returns to 'robert@ubuntu:~/Desktop\$' with a green cursor.

```
robert@ubuntu:~/Desktop$ ./eir -m -f csis.png -n 086 [redacted]
logging in to eir.ie...
login successful
sending webtext (1 of 1)...
webtext sent
Remaining texts: National: 255/300
International: 100/100
Overall: 355/400
robert@ubuntu:~/Desktop$
```

Figure 8: screenshot from tool showing an image being sent

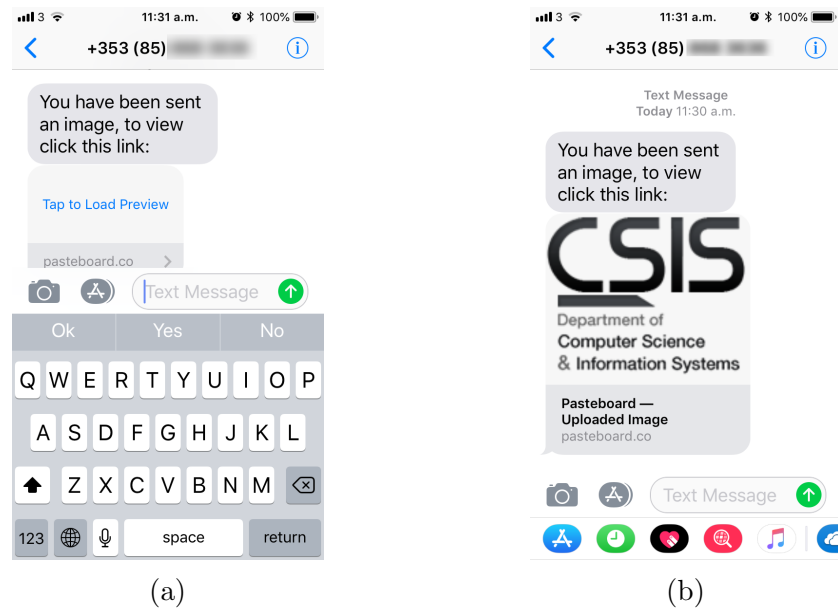


Figure 9: Screenshots from iOS showing an image message being received and viewed with the SMS app

## 4.7 Pasteboard

Initially, the idea was to implement image sending by uploading the image file to a website called Pastebin. This website allows users to upload text, images etc. for the purpose of sharing. Pastebin even offers an API for uploading resources, making it easy to incorporate into tools such as this one. The problem with using Pastebin is that uploading images, and using their API requires a paid PRO account. The website also adds CAPTCHAs for non-PRO users making it impossible to incorporate into this project without a paid subscription.

There are, however, many alternatives to Pastebin. Pasteboard is a much simpler website that allows only images to be uploaded. It is entirely free to use. This website was what I chose to use for this project. Because Pasteboard does not explicitly provide an API, I had to use a similar approach

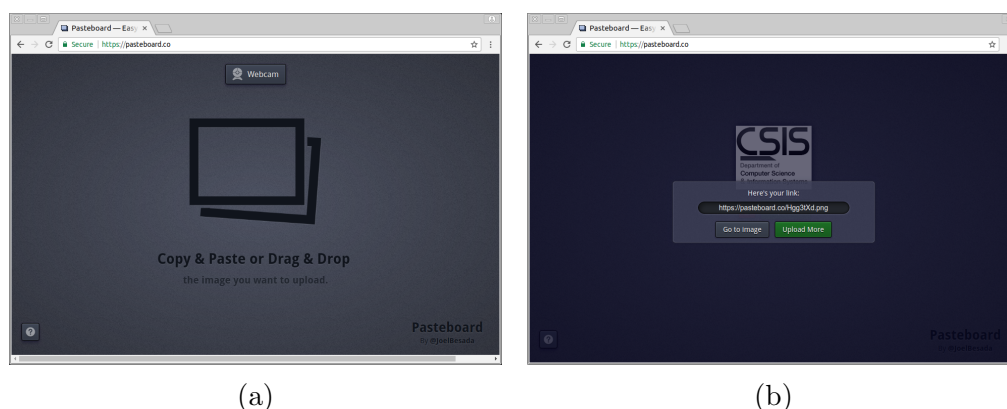


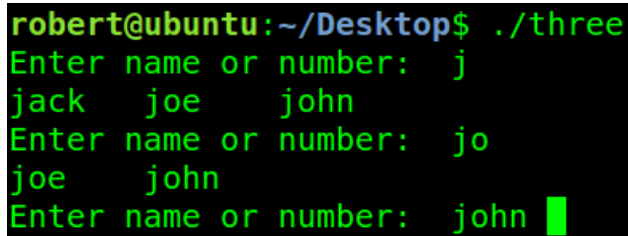
Figure 10: Screenshots from pasteboard.com showing how a file can be uploaded

to the one used on Three and eir's websites. Like before, I studied the HTTP requests being sent and replicated them using `Python`. However, unlike before, the HTTP requests involved uploading a file.

## 4.8 Autocomplete and Comma-separated list of recipients

This tool allows users to save contacts in a configuration file and send messages using an alias rather than typing a telephone number. When entering this alias, this tool supports tab autocomplete. This is implemented using the completion functions in the `Python readline` module.

One simple feature that I hoped to implement was the ability for users to enter a list of recipients, separated by commas. This would make it easier for the user to send messages to multiple people at once. This feature would be relatively easy to implement - simply split the input string where the commas appear, then iterate through the individual numbers. On its own, this feature would not be a problem. However, another feature that I wanted



```
robert@ubuntu:~/Desktop$ ./three
Enter name or number: j
jack    joe    john
Enter name or number: jo
joe     john
Enter name or number: john
```

Figure 11: screenshot from tool showing autocomplete suggestions

to include in the tool was tab autocompletion of contact names as the user enters them. I managed to successfully implement auto-completion, however, it only worked with individual names and not lists. I felt that autocompletion was a more interesting and more important feature to include and therefore I choose to include that rather than comma-separated contact lists.

## 4.9 Changes to Three's website

On one occasion during the development of the project, Three made some changes to their website. This obviously had an impact on the project since the tool is so dependent on Three's system. Soon after I first managed to successfully send webtexts using `Python`, Three updated their website. This update changed the layout and style of the website but also changed the internal behaviour and the format of the `HTTP` requests involved with the webtext sending process. As a result of this update, the script that I had developed to work with Three's website no longer worked. In order to fix this issue, I had to re-examine Three's website, using a similar process to

the one I used to implement the tool in the first place. Although I expected this to be a major problem, I found that it was relatively easy to fix. Once I had updated the HTTP requests in the `Python` script to match Three's new website, I was able to successfully send webtexts again.

## 5 Analysis

### 5.1 Comparison Between this Tool and Existing Websites

In order to understand how successful the implementation of this tool is, it is important to compare it to the existing websites provided by networks such as Three and eir. The purpose of this tool is to provide a reasonable alternative to the existing websites.

The most important feature of this tool is that it can successfully, and reliably, send webtexts. There are of course situations where this is not possible, for example, if there is no internet connection, or if the network's website is experiencing problems. In these situations, both this tool and the existing websites would be affected equally. The only situation where the website would work but the tool would not would be if the website had been changed, but the tool had not yet been updated to reflect this change. This issue is quite rare and is relatively quick to fix.

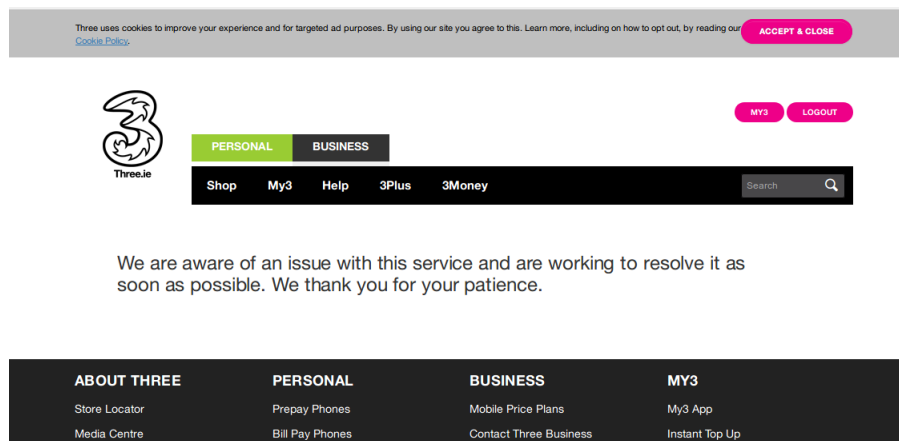


Figure 12: screenshot from three.ie showing an instance when the webtexting service was not working

The existing websites also offer a variety of other features. They allow the user to store contacts so that they do not need to enter a phone number in order to send a message. They also display the remaining webtexts available for a user. Both these features have been implemented in this project.

There are many features in this tool that are not already available from the existing websites. When sending a message using the websites, both Three and eir limit the number of characters that can be entered. It is important to note that this limit is not the same as the maximum size of an SMS message. When using the command-line tool, there is no limit. The `Python` script will automatically split the message into chunks small enough to send over the network's website.

The most important feature that this tool adds is the ability to send images over webtext. In order to send an image, the user supplies the name of the image file. The tool then automatically uploads this image file to a location online and sends the URL to the recipient. While it is technically possible to do this process manually using the existing websites, it is much more tedious, and the websites do not provide any features that would make it easier.

This tool is more efficient in terms of the number of HTTP requests used during the webtexting process. When using the website, resources such as images, CSS pages and JavaScript files are requested. There may be HTML, XML or JSON files requested than contain information that is not important for the process such as account details, welcome messages and offers. The command-line tool only sends the requests that are absolutely necessary for the process such as logging in and sending a message. There are no implicit requests made. For example, if an HTML page refers to a CSS page, the Python script will not load that CSS page. This ultimately makes the tool more lightweight.

From a usability point of view, this tool is faster to use than the websites offered by the networks. To send a webtext through the website, a user must navigate through a series of login screens and dashboards. This tool

automates the login process, as well as the webtext sending process and requires just two pieces of information from the user: the message and the recipient. The amount of user interaction required is kept to a minimum.

Table 1 gives a brief overview of the different webtexting methods. The measurement for the time to send a message is not very precise. The figures shown represent the time taken to send a short message using each method. These figures would change depending on the length of the message, the speed of the internet connection and the level of familiarity of the user with each particular method. For the command-line tool, the interactive time refers to the overall time taken for a user to send a webtext using the tool interactively (being prompted for each input). The automated times refer to the overall time that the tool takes to send a webtext when used in an automated fashion. In this case, the input is passed in the form of command line arguments. The times given for this mode do not include entering the command in the terminal, as they are intended to reflect the behaviour of the tool when embedded in some kind of automated process. Figure 13 shows the times being measured using the Unix time command.

Table 1: Comparison of Webtexting Methods

Comparison of Webtexting Methods			
	three.ie	eir.ie	Command-line tool
Image support	no	no	yes
Time to send a message	27 seconds	39 seconds	interactive: 10 seconds eir - automated: 1.481 seconds Three - automated: 7.336 seconds
Number of HTTP requests used	232	146	2-4
Character limit	768	480	Auto-splits messages



```
robert@ubuntu:~/Desktop/Product$ time ./eir -t "hello world" -n me
logging in to eir.ie...
login successful
sending webtext (1 of 1)...
webtext sent
Remaining texts: National: 253/300
International: 100/100
Overall: 353/400

real    0m1.481s
user    0m0.519s
sys     0m0.072s
robert@ubuntu:~/Desktop/Product$ time ./three -t "hello world" -n me
logging in to three.ie...
login successful
sending webtext (1 of 1)...
webtext sent
Remaining texts: 444/500

real    0m7.336s
user    0m0.653s
sys     0m0.090s
```

Figure 13: screenshot showing time being measured for each network

## 6 Conclusion

### 6.1 Using Python

This project was implemented using `Python`. It was my first experience implementing any major project using this programming language. Overall, I think that `Python` was the best choice for this type of project.

One of the main things I noticed when using `Python` was the amount it differed from more traditional languages such as `Java` or `C++`. Whereas `Java` and `C++` are statically typed, `Python` uses dynamic typing. In some ways, this makes it easier to use. When writing `Python` code, you do not need to know the names of the types you are using. This can be especially useful when you are using a lot of third-party libraries. For example, in the following code:

```
response = requests.post(url)
print(response.text)
```

The response object is returned from a method and used later in the code, however, I do not need to know the name of the object's type in order to write this code. This can make writing code much faster.

The fact that `Python` is dynamically typed did present some challenges. This is mainly because it can be difficult to know how some of the principles of Object Oriented Programming etc. should be applied to `Python`. I encountered this problem before when switching between `C++` and `Java`, for example, `C++` allows multiple inheritance, while `Java` doesn't. However, this problem was much more apparent when using `Python`, due to the degree to which it differs from `Java` and `C++`. The fact that `Python` is interpreted rather than compiled presented a similar challenge, but to a lesser extent. Ultimately, while `Python` code is probably easier to write than `Java` or `C++`, it does take

some time to become familiar with its differences.

## 6.2 Future Work

While the key objectives of the project have been reached, I believe that there is still room for more features and improvements to be added to the tool. There are several small but useful features that could still be added, for example, the ability to create recipient groups in the config file, rather than just individual contacts. There is also the possibility of adding support for more networks in the future, provided that there are no CAPTCHAs in use on their website (this makes it impossible to support Vodafone).

There is also room for this tool to be integrated into other projects. As part of the implementation of this project, I have created a networks module for `Python`. This module is completely independent of the user interface and configuration file etc. and could be used to allow other `Python` scripts to send webtexts. I have also created a `Python` module for uploading images to pasteboard. Again, this module could be used by another `Python` script. It is also possible for a bash script to call this tool, and use command line arguments to pass parameters such as message text.

Another possible feature that could be added to the tool is the ability to schedule webtexts. Although this could be implemented using a separate tool that calls the webtexting tool, there is no reason why it couldn't be incorporated into the webtexting tool itself. This feature is already offered by some of the networks including Three. When implementing this feature, Three's website could be used or, alternatively, the scheduling could be done independently in order to provide greater flexibility. For example, Three's scheduling feature is limited to seven days in the future. If this tool implemented scheduling independently, this limitation could be bypassed.

## 7 References

Von Ahn, L., Blum, M. and Langford, J., 2004. Telling humans and computers apart automatically. *Communications of the ACM*, 47(2), pp.56-60.

BotDetect CAPTCHA Demo - Features. 2018. BotDetect CAPTCHA Demo - Features. [ONLINE] Available at: <https://captcha.com/demos/features/captcha-demo.aspx>. [Accessed 12 April 2018].

ReCAPTCHA demo. 2018. ReCAPTCHA demo. [ONLINE] Available at: <https://www.google.com/recaptcha/api2/demo>. [Accessed 12 April 2018].

## 8 Bibliography

Requests: HTTP for Humans Requests 2.18.4 documentation. 2018. Requests: HTTP for Humans Requests 2.18.4 documentation. [ONLINE] Available at: <http://docs.python-requests.org/en/master/>. [Accessed 29 March 2018].

Three Ireland. 2018. 3G & 4G Mobile Phones and Broadband — three.ie. [ONLINE] Available at: <http://www.three.ie/>. [Accessed 29 March 2018].

eir. 2018. Broadband, Phone, TV and Mobile. [ONLINE] Available at: <https://www.eir.ie/>. [Accessed 29 March 2018].

Beautiful Soup Documentation Beautiful Soup 4.4.0 documentation. 2018. Beautiful Soup Documentation Beautiful Soup 4.4.0 documentation. [ONLINE] Available at: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>. [Accessed 29 March 2018].

14.2. configparser Configuration file parser python 3.6.5 documentation. 2018. 14.2. configparser Configuration file parser python 3.6.5 documentation. [ONLINE] Available at: <https://docs.python.org/3.6/library/configparser.html>. [Accessed 29 March 2018].

Welcome to cx\_Freeze's documentation! cx\_Freeze 6.0b1 documentation. 2018. Welcome to cx\_Freeze's documentation! cx\_Freeze 6.0b1 documentation. [ONLINE] Available at: <http://cx-freeze.readthedocs.io/en/latest/>. [Accessed 29 March 2018].

FrontPage - py2exe.org. 2018. FrontPage - py2exe.org. [ONLINE] Available at: <http://www.py2exe.org/>. [Accessed 29 March 2018].

Welcome to PyInstaller official website PyInstaller. 2018. Welcome to PyInstaller official website PyInstaller. [ONLINE] Available at: <https://www.pyinstaller.org/>. [Accessed 29 March 2018].

Pastebin. 2018. Pastebin.com - #1 paste tool since 2002!. [ONLINE] Available at: <https://pastebin.com/>. [Accessed 29 March 2018].

Pasteboard Easy Image Uploads. 2018. Pasteboard Easy Image Uploads. [ONLINE] Available at: <https://pasteboard.co/>. [Accessed 29 March 2018].

3Community. 2018. Jellysms and webtext - 3Community - 758873. [ONLINE] Available at: <https://community.three.ie/t5/Prepay/Jellysms-and-webtext/td-p/758873>. [Accessed 10 April 2018].

36.1. optparse Parser for command line options python 3.6.5 documentation. 2018. 36.1. optparse Parser for command line options python 3.6.5 documentation. [ONLINE] Available at: <https://docs.python.org/3.6/library/optparse.html>. [Accessed 04 April 2018].

6.7. readline GNU readline interface python 3.6.5 documentation. 2018. 6.7. readline GNU readline interface python 3.6.5 documentation. [ONLINE] Available at: <https://docs.python.org/3.6/library/readline.html>. [Accessed 09 April 2018].