Software Engineering Group Project

Group 6 Maintenance Manual

|  |  |
| --- | --- |
| Author: | Joshua Rogers [jor69] |
| Config Ref: | ManualGroup6 |
| Date: | 27th April 2020 |
| Version: | 1.0 |
| Status: | Release |

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Copyright © Aberystwyth University 2020

CONTENTS

CONTENTS 2

1. Introduction 3

1.1 Purpose of this Document 3

1.2 Scope 3

1.3 Objectives 3

2. Program Description 3

3. Program Structure 3

3.1 Design of the program 3

3.2 List of program modules and their methods 4

4. Algorithms 4

5. The main data areas 4

6. Files 4

7. Interfaces 4

8. Suggestions for improvements 4

9. Things to watch for when making changes 4

10. Physical limitations of the program 4

11. Rebuilding and testing 5

REFERENCES 5

# Introduction

## Purpose of this Document

This document is a maintenance manual, designed to provide program maintainers with answers to specific likely questions they may have and/or show installers/maintainers of the program which part of the program source is most likely to provide the answer.

## Scope

This document covers the overall structure of the program, including design, algorithms and the main data areas. It is designed to assist installers/maintainers of the program answer any questions they may have regarding the software. As such, it should be read by any/all installers and/or maintainers of the software.

## Objectives

The objective of this document is to provide answers to likely questions installers/maintainers of the software may have. It should assist installers/maintainers in:

* Debugging certain issues with installation/configuration of the program
* Extending the program (i.e. modifications to the programs basic functionality)
* Understanding how the program works (e.g. which routines in the program call which other routines)

# Program Description

The Welsh Learner’s program is designed to assist English speakers who want to learn Welsh with learning/revising the Welsh language. The program stores and modifies an internal dictionary of words, comprised of English and Welsh meanings as well as the word type, saved via reading/writing data from/to a JSON file, named dictionary.json. It uses Jackson[1] to achieve this.

The program has a practice list facility, which allows the users to ‘favourite’ words, which will then appear in the practice list. This practice list is stored in a JSON file named ‘practicelist.json’. This is also achieved using Jackson[1]. The user can then either remove words or perform one of a number of revision tests on the practice list words. These revision tests are as follows, given in the requirements specification[2]:

* Flashcards - The learner should be able to be shown either the English or Welsh version of the words on the practice list one by one, and try to remember the translation of the word before being shown it.
* Being given the word in one language and a list of 6 possible meanings in the other language, so that they can select the correct word (the other words should be chosen randomly from the dictionary).
* Being given the word in one language and asked to type in the word in the other language.
* Being given four words in one language from the practice list (only if there are at least 4 words in the list!), and the four meanings of the words jumbled up, and having to match the correct words.
* A random set of the tests in FR9 that should give the user a running score of how well they are doing.

The program gives users the ability to add new words to the dictionary manually. It also allows for the user to switch between ordering the dictionary list of words alphabetically in Welsh or in English. Finally, the program incorporates live search, whereby the list of words gets reduced as the user types based upon matching letters in the list of words.

# Program Structure

The program is made up of several modules to facilitate the many functions of the software. A full list of the modules included in this program and information about the various methods they contain is available in the Design Specification[4].

Many of these modules call routines from other modules.The first major part of the program is the ability for the user to add words from the dictionary to their practice list. They can then perform revision tests on these words. Another major part is the ability for the user to add new words to the dictionary. The third major part is the ability to perform revision tests on the words in the practice list. There are flashcards, 3 different types of test (being given the a word in one language and 6 potential meanings in the other language and the user must select the correct one, being given a word in one language and the user must type in the meaning of the word in the other language and being given four words in one language and the four meanings of them in the other language in a random order and the user must match the pairs) and a test that takes a random set of these tests and gives the user a running score as they complete the questions.

Further information regarding control of flow can be found in the Design Specification[4].

# Algorithms

This section will provide basic information about the major algorithms used in the program. Further information on this topic can be found in the Design Specification[4].

One of the main algorithms in this program is the live search algorithm. This algorithm facilitates the ability of the program to reduce the displayed dictionary list of words as the user types input into the search box, displaying only words that begin with the user input. This covers FR3[2].

The other main algorithms in the program concern the revision test facility. The flashcards algorithm takes all of the words from the practice list, and shows them to the user in order, randomising the language for each one. Clicking the ‘translate’ button shows the user the meaning of the given word in the other language.

The matching test takes four words at random from the practice list and presents the user with them, along with the four meanings in the other language in a different order. The user must them match the pairs, with the buttons turning a specific colour depending on the pair being matched.

The typing test algorithm takes the words from the practice list and presents them one by one in a random language, with the user having to type in the meaning in the other language.

The final implemented algorithm is the choice test. This takes a word from the practice list in a random language and presents it to the user. They are also given six possible meanings and must choose the correct one. If they succeed, they will be notified that they have selected the right answer, if not they will be notified they have selected the wrong answer and told what the correct answer is.

# The main data areas

This section will provide basic information about the main data areas used in the program. Further information on this topic can be found in the Design Specification[4].

The program utilises several different types of data structure as main data areas, to store information such as the dictionary, practice list, etc.

The first of these main data areas is the dictionary. This data area makes use of a HashMultiMap<A, B>, where A is a key of type String, and B is the associated value of type Word. This data type is used to allow the words read in from dictionary.json to be stored internally for display, etc. This data structure is imported from the package ‘com.google.common.collect.HashMultiMap’.

The next main data area is the practice list. This data area also makes use of a HashMultiMap identical to that used for the Dictionary.

Another data area is the internal area used by the ‘JSONProcessor’ class. This data area also makes use of a HashMultiMap as mentioned above, in exactly the same way, named ‘words’.

# Files

The program assumes that in the current directory (the directory containing the program) there is a file named ‘dictionary.json’. The program also assumes that there is a second JSON file, named ‘practicelist.json’, in the current directory. These files are assumed to be JSON files containing JSON information that the program reads in and stores internally whilst running.

The files in the current directory with these names will be read by the program, so it is important to ensure that the correct files, named correctly, are present here.

These are the only files the program will read in, so they must contain the correct data. If either file contains incorrect data, this will still be read by the program and will cause issues. This should be avoided as any incorrect data will then be saved back over the original files.

Upon exiting the program, the internally stored data, including any new words that have been added, etc., will be saved to the same file they were originally read from, dictionary.json in the current directory. The practice list words will also be saved to practicelist.json in the current directory. Any existing data in either of these files will be overwritten with the new saved data and thus lost. If files with these names do not exist, for example they were deleted whilst the program was running, new files will be created with the same names and the internally stored program data will be saved to these files.

# Interfaces

The Welsh Learners program does not make use of any external sensors. The only devices that the program uses are the keyboard, mouse, and display output, and as such the installer/maintainer of this software should ensure that at least one working instance of each of these devices are present in the system running the program in order to facilitate user input and display of output via the graphical user interface provided in the software.

It may be possible for an emulated software keyboard to work in place of a physical device, but this is not recommended and may not allow the program to function correctly.

It is also required that systems running this software have an up to date version of Java[3] and JavaFX installed and working to allow the program to run. This program is designed to run with Java 1.8 or newer. Running an older version of Java is not recommended as the software is not designed to work with this and may not function correctly as a result. If no Java and/or JavaFX installations are present then the program will not be able to run on the system.

# Suggestions for improvements

There are several suggestions for improvements that could be made to this program. Unfortunately due to time constraints, some features were not implemented. The first of these is the ability to run a test that presents the user with a set of tests that is a combination of the 3 different test types and keeps a running score. This requirement is known as FR10[2]. We were not able to include this feature, but it could be added by creating new JavaFX classes for this part of the program.

Another suggestion for improvement for this program would be to choose a JSON library and use this for all JSON functionality. The program makes use of Jackson[1] for writing data when saving, but utilises a different JSON library to read in the JSON data from the dictionary.json and practicelist.json files. This is a potential weak point of the program and could lead to issues in the future. The program uses multiple JSON implementations as multiple people have been working on the code, and as it has evolved over time it has become more and more difficult to fix this issue. Due to time constraints we have been unable to resolve this problem.

# Things to watch for when making changes

Certain aspects of the program are ‘fragile’, meaning that if they are not correctly used or altered by an installer/maintainer they can have a knock-on effect and cause issues with the rest of the program.

The most important thing to watch for is regarding the JSON files. If these files are formatted incorrectly/contain incorrect or missing information this can cause several issues with the program. It may result in words having blank entries for English/Welsh meanings or word types, for example. This will cause issues with revision tests.

Another possible issue this could cause is that the program could crash upon startup if the JSON data is formatted incorrectly. If the JSON file is for example missing it’s closing square bracket, the JSON implementation we have used may have issues reading the file and will likely throw an exception, causing the program to crash. This type of exception is not handled by the code as having an incorrectly formatted JSON file is a serious issue, so we feel the program should not work if the file is incorrectly formatted, until the issue is fixed. In the case of an incorrectly formatted JSON file, it is advised to use the ResetDictionary.bat maintenance tool mentioned in section *11. Rebuilding and Testing*. This will replace the current dictionary file with an original, unedited copy with correct formatting.

# Physical limitations of the program

This program does not require a large amount of CPU resources due to its relatively simple nature. The most intensive aspect of this program is its memory usage. Due to the number of words present in the dictionary.json file, all of which are read into the program and each have some memory allocated to them, the program can typically use between 1 and 1.5 gigabytes of RAM. Systems with less than 2 gigabytes of RAM are not recommended to run this software with the full dictionary. It is possible for installers/maintainers of the software to manually edit dictionary.json and remove some of the words in order to reduce the system memory requirements of this program. This is not recommended unless the individual is experienced with JSON and feels confident in their ability to edit this file correctly. An incorrectly formatted file can result in issues as described in section *6. Files.*

Based upon the resources used by the program in testing, the recommended minimum system requirements for this program are as follows:

* CPU: 1 gigahertz single core
* RAM: 2 gigabytes (for full dictionary)
* Storage space: 1 gigabyte disk space
* Display output
* Keyboard
* Mouse

These requirements are estimates and may not be precisely accurate. Whilst it is potentially possible for the program to run on systems with less CPU/RAM/Storage space than listed above, this is not recommended as the program may not function correctly. Display output, keyboard and mouse are all required to ensure user input and output of information by the program via the graphical user interface are facilitated.

# Rebuilding and testing

To rebuild the program, the provided IntelliJ project must be opened in JetBrains IntelliJ IDEA[5]. The project is located in the ‘SRC\Source\_Code’ directory of the repository. From here, the user must select ‘File’ from the top menu, then choose the ‘Project Structure’ option from the drop-down menu. The user must then select ‘Libraries’ option and ensure that the Maven dependency for Jackson is present. If it is not one should be added, using the ‘+’ icon, for ‘com.fasterxml.jackson.core:jackson-databind:2.10.3’.

Following this, the user must select the ‘Build’ option on the top menu bar, the select the ‘Build project’ option from the drop-down list. A shortcut for this can be accessed by using the key combination CTRL + F9.

From here, to run the program the user can either select the ‘Run’ top menu option and then click on the first option from the drop-down list or use the key combination Shift + F10. The user has now built the program and it should now be running.

Maintainers can find the tests we used to test the program located within the IntelliJ project. These are JUnit[6] tests, so the system must have a working version of JUnit installed and added to the project (using the same method as described above for verifying the presence of the Jackson Maven dependency. New tests can be added as/when necessary by editing the JUnit files provided and adding new tests.

To run these tests simply use the same build instructions given above, but instead run by right-clicking on the JUnit test class name and selecting ‘Run’ from the menu that appears. This will then tell the user which tests have passed/failed.

All documents are written using Microsoft Office[7] software, including Word, PowerPoint and Excel. These can be viewed using their relevant software, as well as a range of open source, third party tools. It is not necessary to rebuild these documents. Editing them is not recommended.

Maintenance tools are available to assist with rebuilding the program. They take the form of .bat files (batch scripts) that should be run when needed. They should not be moved from their locations as doing so may cause the script to not work or delete files unintentionally. The scripts are: ClearPracticeList.bat, which is used to clear the practicelist.json file, and ResetDictionary.bat, which is used to replace the dictionary.json file with an original, unedited copy. Both scripts are located in the ‘SRC\Maintenance\_Tools’ directory of the repository. These tools are only compatible with Windows. They are also located in the first folder level of the .zip file containing the code.

REFERENCES

[1] GitHub. 2008 – present. FasterXML/jackson. [ONLINE] Available at: https://github.com/FasterXML/jackson. [Accessed 29 April 2020]

[2] Software Engineering Group Projects: Welsh Vocabulary Tutor Requirements Specification. C. J. Price. SE.QA.CSRS. 1.1 Release

[3] Oracle. 2020. Java. [ONLINE] Available at: https://java.com/en/. [Accessed 29 April 2020]

[4] Located in the ‘DOCS\Architecture’ directory of the Group 6 GitLab repository

[5] JetBrains. 2000 – present. IntelliJ IDEA. [ONLINE] Available at: https://www.jetbrains.com/idea/. [Accessed 30 April 2020]

[6] JUnit. 2020. JUnit 5. [ONLINE] Available at: https://junit.org/junit5/. [Accessed 30 April 2020]

[7] Microsoft. 1990 – present. Office 365. [ONLINE] Available at: https://office.com. [Accessed 30 April 2020]**DOCUMENT HISTORY**

| *Version* | *CCF No.* | *Date* | *Changes made to document* | *Changed by* |
| --- | --- | --- | --- | --- |
| 1.0 | N/A | 27/04/20 | N/A - original version | JOR69 |