AUTOMATIC NAME GENERATOR WITH EXTERNAL SOURCE FILES AND EDITING SYSTEM

Cameron Hillier

Written as a summary, discussion and explanation of the above application

Farnborough, Hampshire

September 2019

DECLARATION

This report serves no meaningful purpose, it was written solely for the sake of practice by the writer. Reading, understanding and interpreting anything of value from this document (of which the writer believes there is none) is left as a (likely fruitless) exercise to the reader.

This report is not intended for any kind of public display and is written in the full expectation that no one besides the writer will ever read it.

Abstract

This report discusses the creation and development of the Automatic name generator application, its associated source files and the integrated editing system. The initial scope of the project, how it was expanded and how scope creep was avoided, as well as the previously implied new features are also discussed.

In addition, this report goes over cancelled and possible future plans for the application concept. Finally, this document also examines the testing process for the application and verifies the functionality of many of the product’s features before reaching a conclusion on the project as a whole.

Table of contents

Chapter 1 – Introduction and methodology

Page 5

Chapter 2 – Initial system, design and functionality

Page 7

Chapter 3 – Rejected ideas and possible future expansion

Page 9

Chapter 4 – Testing and evaluation

Page 12

Chapter 5 – Conclusion

Page 16

Appendices

Page 17

End of Preamble

Chapter 1

Introduction and methodology

To begin this report, it makes sense to start at the beginning, as such this chapter focuses on defining the project by setting out its objectives and outcomes and discussing the rationale and methodology that underpin the project as a whole.

* 1. Defining the scope of the project

The scope of the project is small, simple and straightforward, but with a defined path of further growth should the opportunity to expand present itself. The initial scope of the project was, ostensibly, what it said on the tin, a name generator that functions by selecting a random entry from a list of prefixes and a list of suffixes and combining the two to produce a name.

The initial scope, however also had a view to expand, with several possible additional features planed, these included, in no particular order: name history, external source files, a source file editing system(dependant on the previous feature), additional generators and many smaller items, such as a main menu to allow switching between these options, that have largely gone unremembered.

As can be surmised, the scope of the project was deliberately designed to control complexity by starting simple and adding more individual features as the project progressed, allowing it to become fully featured without as much groundwork and early development planning as would otherwise be required.

1.2 Outcomes and objectives

The objectives of the application were as follows:

1. To produce functional name generator as described above
2. Modification of the generator to allow the use of external source lists
3. The implementation of a method to switch between multiple generators
4. Creation of a system for viewing the saved name histories within the application
5. Development and integration of an editing system for the source files from within the application itself
6. Addition of further name generators as the developer sees fit.

(note: all other objectives were considered secondary to the first.)

The primary outcomes of the project were:

1. The name generator application
2. The requisite source files
3. The integrated editing system for said files
4. This report detailing the application and its development
   1. Rationale

The rationale behind this project was relatively simple, the creator wished to further practice application development in the C# programming language using the visual studio development environment, the project concept was intentionally kept simple to prevent it from spiralling out of control and becoming unnecessarily complex.

The primary reason for wanting to keep the concept simple was that a previous project, a mock banking system (in reality a SQL database manipulation program) had been severely compromised by feature creep at the planning stage. It became convoluted with too many dependencies and interlocking features required for it to function at all that it would have to be dramatically cut down and re-implemented step by step in order to be completed.

To avoid repeating those mistakes this project was handled differently, a simple, core feature, the name generator would be created, with additional features added on or integrated one at a time after the previous one was complete. This approach avoided having to balance too much at once, whilst still allowing a fully featured product to be developed.

An additional, unplanned bonus was that due to lack of interdependence and common code the adding of new name generators and features became relatively simple. The writer acknowledges that this is less well optimised than may otherwise be possible but feels that this is a worthy trade off as it makes little difference at this scale and the gains in ease of development are substantial.

* 1. Summary

The purpose of this project was to produce a name generator application and an editing tool for its source components. The project was undertaken in order to practice development in the visual studio environment whilst using a different methodology to previous exercises to implement a relatively simple concept to the point of being a complete project.

End of chapter 1

Chapter 2

Initial system design and functionality

Once the system has been defined and objectives set it is time to get down to details and define. In this chapter the functionality and design of the system is discussed.

2.1 Name generators

The primary outcome of the project was the name generators. The generators function by using a random number generator to select a random index in an array where each index is a different word or name. selections are made from each appropriate array and then combined to create the finished name.

An example would be a personal name generator with an array of first names and one of last names, a random number is generated and the name in the matching index of the first name array is selected, in this case “John”, this is repeated for the last name array and “Smith”. The two results are combined to make the full name “John Smith”, which is written as the result.

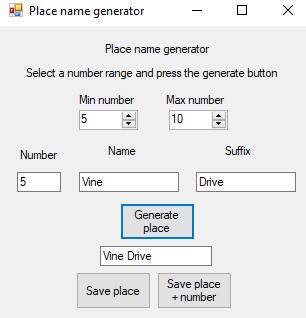
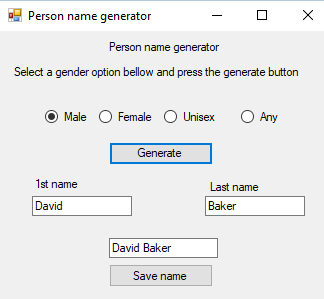
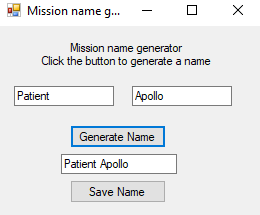
The generators have also been modified to fill the arrays from externally stored files, this was done to clean up the code, make the lists easier to edit, allow access to the files for the editing system and to allow changes to the list to persist between uses of the editing system.

Figure 1: 3 examples of name generators available within the application, showing a mission name, male name and place name having been generated.

2.2 Detailed design

The design of the application is fairly simple, it features a main menu with buttons for opening each form with the 3 generators grouped together at the top and the secondary forms grouped up bellow.

Each of the generators has a fairly normal layout, consisting of a title/description, with additional controls (such as number range or gender selection) grouped together above the centrally located generate button. Labelled boxes are located bellow for both the prefix and suffix as well as a final box for the final result.

The ancillary forms also follow a similar design methodology, with titles and descriptions at the top, with the relevant lists located in the centre with controls grouped up bellow.

The design is based on common design ideas such as grouping similar elements together, with important information at the top and the major elements in the most prominent positions.

Design documents can be found in the appendices of this report (Figure 2).

2.3 Functionality

The primary functional areas of the application are found in two distinct areas, the name generators (the functionality of which has been discussed above) and the source file editing system.

The file editing system functions by loading each source file into an array and displaying each array in a list for the user, the editing form contains all the controls necessary for the user to edit these lists as they desire. Once the user is done editing, they can choose which lists to save, this takes the list, places it in an array and uses that to write the new information to the source file.

In addition, the application also contains ancillary functionality to improve the capabilities of the system. The most notable example is the history system, the user is able to save any name they generated to the relevant history file, the user can view and delete history entries from the view history form.

Further ancillary features include a main menu, generator specific options, such as sex for the person name generator or number for place names and selection and sorting options for various lists.

2.4 Summary

The functionality of the application is broken down into several major parts, each performing a relatively simple and self-contained task or set of tasks. The design is fairly simple, following basic principles with little unnecessary clutter, while maintaining everything required for the successful completion of its goals.

End of chapter 2

Chapter 3

Rejected ideas and possible future expansion

When every project is being developed a wide variety of ideas and possible features are suggested, however, for various reasons not all of them get into the final product. This chapter discusses some of these ideas and why they didn’t make it to the end.

3.1 Rejected ideas and cancelled features

Whilst the project achieved its primary objectives several features were cancelled or ideas rejected for various reasons, some of the more noteworthy examples were:

* Dynamic additions to the edit page
* A modified UI for the edit page
* Addition of a system to the name generators to prevent repeated answers

Dynamic additions to the edit page would have allowed new generators to be added automatically, this would likely have taken the form of a system for automatically adding lists based on detected generators. This idea was rejected because it would have been extremely complicated to implement and could potentially mess up the edit page’s UI or cause errors, all for the sake of a very rare occurrence which would have required user input regardless. A more sensible solution of producing a guide on how to add generators was adopted instead.(A copy of the text of this document can be found in the appendix as figure 3)

The second rejected idea was a modified UI for the edit page, the plan was to reduce the amount of clutter by amalgamating buttons and maybe even lists together to make the page easier to navigate. This idea was rejected because it only provided a marginal improvement in visual clarity but dramatically worsened the user experience and program flow whilst making the editing screen less intuitive to use.

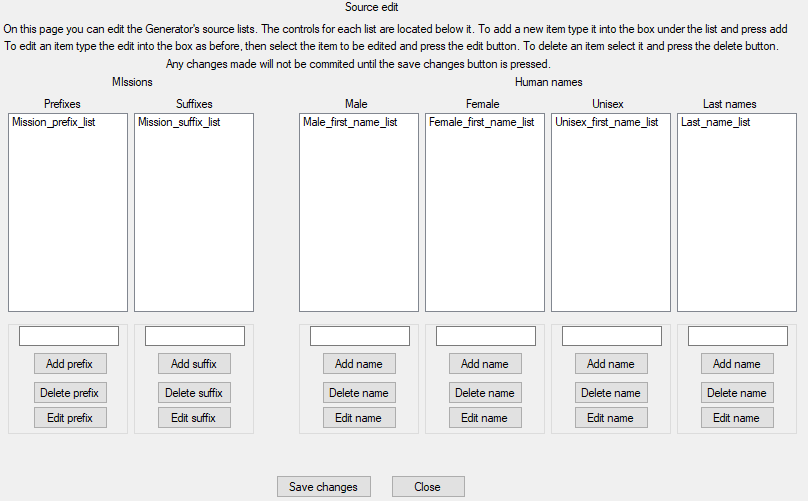
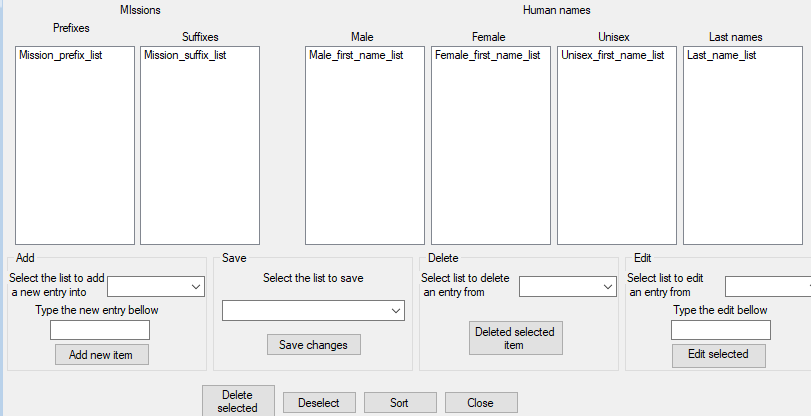


Figure 4: The original edit screen (left) and the failed redesign (right)

The Third and final cancelled feature that will be discussed here was a system to prevent the same name being generated twice in a row. This feature was cancelled because once the source lists were made a decent size the likelihood of repeats occurring decreased exponentially to the point where this feature, whilst still technically useful became redundant and so was cancelled to avoid unnecessarily complicating the application.

3.2 Possible future expansion/modifications

Although the project was successfully completed and achieved both its primary and secondary objectives there is still room for improvement and opportunities to make changes. Such modifications may improve efficiency, add features, make for a more advanced code, or simply provide further polish to the user’s experience.

The proposed future changes we will be discussing are:

* Internal source files
* Subdivision and common method calling
* Relocation of array content collection to form load
* General polish

The first idea, internal source files came as a result of a later project that involved this feature and recognising that this feature could be retrofitted into the application. Implementing this would make the project more portable since the file references would be relative links within the project file rather than absolute paths from the root directory, which would break when attempting to run the app on a different device. It is thought that this would be a relatively simple and worthwhile improvement.

The second modification would be modifying the code with some general improvements to the system. Firstly, common pieces of code (such as the RNG used in the name generators) would be made into public methods to be called in order to improve efficiency and cut down on the redundant code. This was not done initially since the app was deliberately designed to be as simple.

Secondly the array content loading would be moved to the appropriate form load segment rather than the form’s button press action. This modification would make the program more efficient as it would only access the source files once per generator form load instead of for each use. Whilst this is more efficient it would also require a method of updating the arrays should the files change, which may cause substantial complications over this minor optimisation.

The final possible modification that will be discussed here is the simple idea of adding more polish to the system. The process of polishing the application may include steps such as optimising code, tidying up the UI and textboxes and improving the UX. This is a simple but ongoing process that has already been started. The reason for including this here is that many of these possible improvements to the application may be made in the future.

3.3 summary

The first portion of chapter 3 reflects on rejected ideas and why they failed, these were generally attempting to add functionality that fell flat as they added unnecessary complication for a minor feature or represented unacceptable compromises to the user experience.

The second segment discusses possible future ideas and what benefits they may add, as well as why they are not currently implemented. These mostly come in the form of features, code improvements and polish, and have not been implemented mostly because they conflict with the intentionally simple nature of the application.

End of chapter 3

Chapter 4

Testing and evaluation

Once the application is written an important step must be taken before it can be called complete. Testing and evaluation must be carried out to ensure that everything is, in fact working as intended.

4.1 Testing

Testing of the application broadly falls into two categories, unit testing and final integrated testing.

Unit testing mostly involved testing each individual feature and specific aspects of it and its functions as development was ongoing, typically black box testing it immediately after creation. This was done as it can be a fast and efficient way of finding small mistakes or logic errors without the confusion of multiple features and layers of code, helping ensure that each part is functionally correct prior to integration. Most of the testing was done this way.

Integration testing was fairly simple since there was not much interplay between forms, with only the main menu being heavily reliant on other forms. Beside the menu the forms only interacted by updating common text files that the others would read when opened. As a result, testing mostly consisted of ensuring the menu’s functionality and ensuring that the generators would read correctly from the appropriate files, particularly when re-opened in the same run following an update made in a different form.

The following is the test plan for the integration and feature testing of the application.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test no. | Test name | Expected outcome | Actual outcome | Success/failure | notes |
| 1 | Main menu load | Main menu loads | Main menu loaded | Success | Functions as expected |
| 2 | mission gen load | Mission gen loads | Mission gen loaded | Success | These tests include data |
| 3 | place gen load | Place gen loads | Place gen loaded | Success | Data load verified by tests 5,6 and 7 |
| 4 | Person gen load | Person gen loads | Person gen loaded | Success |  |
| 5 | Mission gen function | A name with a prefix + suffix | A mane with a prefix+ suffix | Success | Name was Patient Apollo |
| 6 | Place gen function | A place name with no. 5-10 | A place name with no. 5-10 | Success | Name was 5 Vine Drive |
| 7 | Person gen function | A male name and last name | A male name and last name | Success | Name was David Baker |
| 8 | History load | History form loads | History form loaded | Success | Loaded with all previously saves |
| 9 | History function | Mission name should appear | Mission name on correct list | Success |  |
| 10 | Save function | The name from test 7 in a list | Name showed in person list | Success | The name was David Baker |
| 11 | History delete | Name from above removed | Name was removed | Success | ^ |
| 12 | Edit load | Edit form loads | Edit form loaded | Success |  |
| 13 | Edit add | 2 new prefixes are added | New prefixes in mission list | Success | “Hidden” and “Hopeful” added as a prefix |
| 14 | Edit change | A prefix change | The prefix changed | Success | “Hidden” changed to “Silent” |
| 15 | Edit delete | A prefix is removed | Prefix is removed | Success | “Silent” deleted |
| 16 | Edit update | List updates text file | File updated from list | Success | Prefix list was updated |

Evidence for tests 2-7 can be found in figure 1 (page 7)

Evidence can also be found in figure 4 (page 9) and figure 5 (appendices, page 20)

Testing scenarios were mainly devised around simple tests for each function that would finish with as little as possible (preferably nothing) left changed after the fact. Scenarios included:

* Testing the generators by generating a name with given parameters as appropriate.
* Test of the history function by saving the name “David Baker” that was generated in the previous article, listing it and deleting it.
* The editing form was tested by adding names, editing and then deleting one of them, the source file was then updated.

The results of these test scenarios can be seen in the test plan above.

These test scenarios were used to ensure that the program would function as intended during expected and likely situations, abnormal behaviour testing and adaptation was also conducted but is not documented here due to the ad-hoc nature of the process.

4.2 Evaluation

Evaluation of the application will focus on how well it performed, whether the objectives were met and if the expected outcomes had been delivered, the individual features will each be evaluated, followed by the application as a whole.

Please see chapter 1 section 2 (Pages 5 and 6) for a reminder of the objectives and outcomes.

Firstly, the generators, these were simple and performed largely as expected. They used external text files to function, just as was planned. They featured various options to add variety. The UI was simple and had little clutter but was fairly intuitive, using common design ideas to help the user navigate. Overall it could be said that the generators did their jobs well.

The generators satisfied objectives 1,2 and 6 and outcomes 1 and 2.

The history form was very straightforward as it simply read from the history files and displayed them, with the delete option being a built-in function of C# lists and the save function being a simple version of the one from the edit page. The UI was simple but did what it needed since there really wasn’t much that needed showing. It would be fair to say that this form was fully functional if a little boring and simple.

The history page completed objective 4 and was prerequisite on outcome 2’s completion

The editing screen was the single most complex part of the application. It involved reading and listing all of the source files and controlling lists, like before but multiplied. Selecting which list to save was by far the single most complex piece of code in the project. Once the main functions where complete smaller functions such as sorting or deselecting where added. Whilst each individual step was, by and large simple the quantity and interaction developed the screen into a fairly complicated segment of the application.

Needless to say, having so many lists, each with 3 buttons and a textbox, a save function, several ancillary functions and an explanation made the UI Very busy. Efforts have been made via grouping and ordering to make it easier to follow and reduce user confusion without compromising UX, but it was definitely far from ideal and future changes to this screen would do well to address this issue.

The editing screen fulfilled objective 5 and outcome 3.

Finally, the application itself, this consisted primarily of each of the previously mentioned items(including 3 generators) and a main menu for navigating between them. The app was fairly simple to navigate since the only options were to open a new form from the menu or close the current, non-menu form. The Ui for the app itself is simple, with the menu simply having the generators grouped at the top with the extra functions bellow, making it clear where everything belongs. The general idea of grouping similar items together and keeping it simple is consistent through the entire application.

Overall the application could be said to be highly successful as it completed all of its objectives and delivered its outcomes in a simple, intuitive and easy to use manner. It had a simple UI and kept the UX and ease of use of the program intact.

The application, distinct from the individual elements completed objective 3 and integrated the previous steps as well as completing the integration required in outcome 3.

Careful readers may have noticed that whilst every other outcome has been mentioned, and their completion demonstrated, outcome 4 has been entirely absent, after all it is external to the rest of the project proper. The writer hopes that its fulfilment is self-evident but leaves it to the reader to make their own judgement as to the standard of quality of the final outcome.

End of chapter 4

Chapter 5

Conclusion

With the project completed and fully evaluated all that remains is to comment on the process of development, summarise and conclude the report and the project as a whole.

5.1 conclusion

The process of developing the system was relatively simple and lead to a fairly easy development since each segment was developed separately and combined later through a common set of files, this method proved very quick and effective for this project.

The project was largely successful, having produced an application that completed all it’s objectives in a simple and easy to use manner and generating each of its expected outcomes successfully, completing and refining them to an appropriate degree.

In conclusion the project served well as a programming exercise that did everything it was intended to in terms of allowing certain methods of developing, designing and coding an application to be practiced. The project also managed to produce a useful writing exercise by providing a basis for this report, which has hopefully been of satisfactory quality.

End of chapter 5

Appendices

Name generator source list content origins

Prefixes and suffixes for the mission name generator were taken from the XCOM wiki at:

<https://xcom.fandom.com/wiki/Missions_(XCOM:_Enemy_Unknown)> (21/8/19)

The place name components were found largely by looking around the local area on google earth with some more esoteric or foreign names coming from examples that appeared when googling popular street names. (Early September 2019)

Components for the 4 lists required for the person name generator largely came from the following locations:

Men/women

<https://www.ssa.gov/oact/babynames/decades/century.html>

<http://usefulenglish.ru/vocabulary/mens-names>

<http://usefulenglish.ru/vocabulary/womens-names>

unisex names:

<https://en.wikipedia.org/wiki/Unisex_name#European>

Last names:

<https://en.wikipedia.org/wiki/List_of_most_common_surnames_in_Europe#United_Kingdom>

These were all accessed mid to late September 2019

Figure 2: design documentation

Program flow

User starts at main menu

User choses generator

User selects generator options (if applicable) and presses the generate button

The name is generated

The user then either generate a new name, save the current one, return to the menu or exit

User can then view history, which also allows them to delete contents

The user can also access the edit page from the menu

Each source list can be viewed, edited, added to, deleted from and updated from this list

The user can return from each secondary form and exit the application

Pseudocode

Main menu

Show form BTN (Repeated for each form reachable from the main menu)

Create instance of form x

Display form x

Hide this form

Mission name generator

Import prefixes to prefix array

Import suffixes to suffix array

Generate name BTN

Int rng = new random number

String chosen prefix = prefix array[rng]

String chosen suffix = suffix array[rng]

Prefix display. Text = chosen prefix

Suffix display. Text = chosen suffix

String final name = chosen prefix + chosen suffix

Final name display.text = final name

Save name BTN

Return If final name display.text = empty

using

New streamwriter(file location)

File. WriteLine(final name display.text)/using

Figure 3: How to add a generator document

The following is a copy of the text, in full, of the document “How to add a generator” contained within the project folder for the name generator application.

Creation

1. Add the necessary source lists in the Generator components folder.

2. Add an empty file for list history in the History folder

3. Create a new form for the generator and create the generator.

4. Add save function to the form.

Integration

1. Add a button to the Main menu form to open the form.

2. Add a List for the generator history in the Name history form.

3. Populate the list on load, this can be done by copying and adapting the existing code for the other lists.

4. Create entries for the list in the history deselect and delete buttons, do this as above.

Integration - Edit source page

1. Add lists for each source file.

2. Populate the lists on load, this can be done as above, make sure they are cleared prior to filling.

2. Add controls for each list, port the code for each button as previously mentioned.

3. Open the edit items box in the save select box and add options for each new list.

4. Add a new option to the if statement that finds the right list in the save function, do this for each new list.

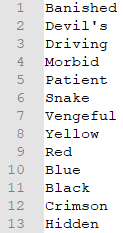
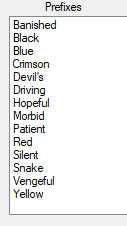
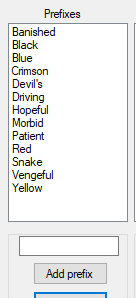
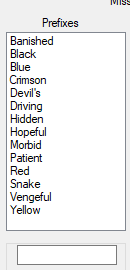
5. Port over the code for each option from other save options.

6. Add each new list to the sort stack in the page load section.

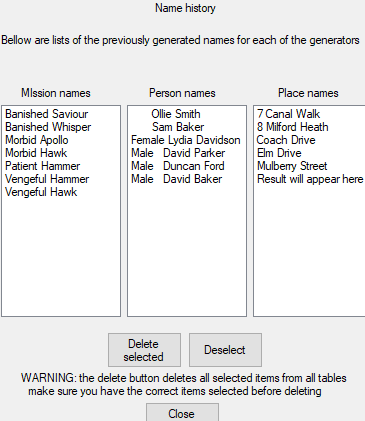
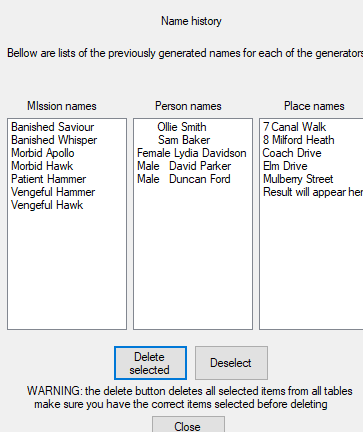
7. Add each new list to the deselect button.

Figure 5

Test evidence screenshots



The above are evidence screenshots for the edit page tests.



The above are evidence screenshots for the history page.

End of report

End of Appendices