Schappler's Snippet Saver – Dynamic Multi-Language Code Library

Nick Schappler

May 7, 2011

1. Abstract

Computer science students are notorious for relying on google while programming because of the quantity and quality of software references available online. My intent with this project was to provide students with a user-friendly, streamlined approach to saving coding tricks they stumble across that may be useful somewhere further down the road. One of my main goals was total flexibility. A user can easily shape his own library by modifying the supported languages or adding/editing/deleting code entries in rich text. Code entries have two attributes, a title and the code itself. As far as accessing code entries, a user can do one of two things. By selecting a programming language, a user is immediately shown an alphabetically sorted list of code entry titles for that language. Browsing through the code entries is extremely fast and straightforward. There is also a multi-word search feature to quickly find a specific entry. Search matches are separated by title or code matches to display results by their relevance. Ultimately, the program can be a great utility that unobtrusively runs in the background while any programmer works. Implementation was in Python using PyQt for GUI development, regular expressions for searching, and list comprehensions for traversing certain data structure elements.

2. Keywords

Snippet: A piece of code, generally part of a larger selection.

Partial Match: Default searching method. Searches for any existence of the search text. For example, waterfall could be returned by a search for water.

Exact Match: Optional searching method. Searches for exact search text. For example, waterfall could not be returned by a search for water.
Regular Expression: A way of representing data using symbols. They are often used within matching, searching or replacement algorithms, especially in strings.

3. Introduction

_Schappler's Snippet Saver_ is a program that will prevent a user from repeatedly searching Google for answers to common problems (or even just tricks-of-the-trade) that they have discovered during their programming development. First, it is an excellent tool for a programmer to use in browsing and searching for an answer to a coding problem they have encountered. There are many useful websites that can be used as programming references. However, once a programmer finds something that resolves his problem, he typically abandons and forgets the solution. This code library serves as an unobtrusive medium to remember these coding solutions. If a user knows that he has previously come across a similar problem and has used Schappler's Snippet Saver while programming, he can reference his previous solution in seconds. The alternative, googling the problem and searching through coding forums, generally takes a great deal more time and effort.

The code library is also an extremely effective study aid for programers who have fallen out of touch with a language. Rather than picking up a 300 page coding book or traversing hundreds of programming websites, a user can browse through his code entries for a particular language and get a feel for some of the common problems he typically faced. Once coders get past a basic introduction to programming, it's often best to look through small code examples to learn the functionalities and features of a language. Hypothetically speaking, let's say a computer scientist uses primarily C, C++, and Python during his collegiate studies but switches to working on a ColdFusion project for a number of years following graduation. He then switches jobs and starts working on operating system development for Microsoft. The computer scientist who uses this library is better equipped to begin programming immediately because he has a diverse collection of easily searchable code snippets available at his disposal. This program helps to lower the learning curve associated with re-familiarizing oneself with a particular programming language.

The main structure of the code library is commonly implemented throughout all of academia and the commercial sector. In _Schappler's Snippet Saver_, there are programming
languages that are associated with code library entries which themselves consist of a title and a code snippet. In the field of Art History, I have seen museums that utilize a similar structure for electronically databasing works of art. From the main structure, a user can select/add/delete an artist. For each artist, the user can add a work of art which minimally consists of a title and a digital image. The program is searchable in a similar manner to that of Schappler's Snippet Saver, but artist names and work titles are searched rather than programming titles and code entries. Additionally, OptionsHouse, an online stock trading broker, uses a similar structure to the code library in presenting recent news to its members for positions they hold. When a user logs into his OptionsHouse account, one of the first things presented to the user are his position, the stocks he actually owns. By selecting a stock, a plethora of news titles populate in a nearby window. From there a user can select a title to read the news article itself. Unlike Schappler's Snippet Saver, the news platform does not allow for searching, but rather sorts by publish date and time.

The structure used for the code library is so widespread because it can be easily implemented, particularly in relational databases. Existence of such libraries typically differ in just a few ways: how many 'layers' of data exist (levels of indirection ie. Languages → Titles → Code Entries), how the data is presented (sorting order), and how it's accessed (browsing, searching, etc).

4. Usage Examples

The image below is the standard layout of Schappler's Snippet Saver when it is first opened.
The GUI is largely divided into two sections, left and right. The left half shows a list of the languages in the code library. Below is the area to add a code snippet. Below that, there is a searching mechanism for filtering entries in the code library. On the righthand side, there is a graphical display of the code entry titles for that selected language. Below are buttons for viewing or deleting code entries. Clicking another language dynamically changes the code library on the right to display the entries for the newly selected language. Note that both the programming languages and the entries in the code library are ordered alphabetically for viewing ease.

To add a programming language to the code library, simply click on the *Add Language* button. When clicked, the user will be prompted with a modal dialog requesting the new language name.

After entering the language name, a user can accept the action by clicking *OK* or cancel by clicking *Cancel*. If accepted, the language will automatically be added to the code library. To remove a language (and all of its code entries), select a language and click the *Remove* button. Similarly, a user can accept this action by clicking *Yes*, or clicking *Cancel* to cancel.
Now, let's discuss how to access entries of the code library. Select the programming language and the code entry of interest then press the View Code button. The code entry will be brought up as a separate dialog.

The window title of the dialog is the code entry title for readability. The code library is capable of processing rich text, so you can markup your entries to make them more presentable (as shown in the above image). The text is non-editable by default, but a user can change this by selecting the Edit button. After editing the entry, a user can click the Save button to update the code library. Viewing code entries uses non-modal dialogs, so a user can view as many entries as he wishes while still interacting with the main code library, or any other program for that matter.

A user can easily add a snippet to the code library for the selected language by clicking the Add a Snippet button. A dialog is brought up and a user can name the code entry with a title and paste their code entry. As before, you can accept the action by pressing Add Snippet, or cancel by pressing Cancel. If added, the code library will automatically be updated to show the new addition.
Now for the search feature. A user can filter the code entries of a selected language by typing their word of interest into the search line and pressing the Search button. A list of results will be shown in the code library on the righthand side. Results matching the search in their code entry titles will be bolded and placed near the top since they are likely of highest relevance. Results that have matches in their code snippets are shown in regular font. See the example below.

Note that searching for the word 'bash' returned numerous results, some that matched in the titles (bolded) and others that matched in the code entry itself. By default, the search is not exact match. You'll notice how the second bolded result in the above image contains the word 'bashability'. If you wish to only view results that exactly match your search criteria, check the Exact Match checkbox to the right of the Search button.
Additionally, the search feature accommodates multi-word searches. The search can also be cleared by the Clear button. If pressed, the search field will be cleared, and the code library will once again return to showing all of the selected language's code entries, sorted alphabetically.

Lastly, there are two main shortcuts that improve user interaction, specifically in speeding up all library processes.

- Double clicking a language = Pressing the Add a Snippet button
- Double clicking an entry in the code library = Pressing the View Code button

5. Approach and Methods

Schappler's Snippet Saver was implemented using python with PyQt. The program, in its entirety, consists of 4 python files. Different files contained python classes separated by their interaction with the overall program. There is a main window python file with various PyQt workings and the search feature. The remaining three files are for language management, operations, and library management. The language management file contains two classes for adding a language and removing a language, respectively. The operations file consists of only one class for adding a snippet to the library. Lastly, the library management file contains two classes for viewing a code entry and deleting a code entry, respectively. The goal of this segregation was to make future development more convenient.

The program consists of mainly one data structure, a list of of two-tuples. The tuples are pairings of a string (the language name) and a dictionary of key:value pairs for that language.
The keys are that language's code entry titles and the values are HTML formatted code entries. The benefits of having one large data structure was centrality, particularly as it relates to the relative ease of saving and loading library data. However, there were situations where maintaining a few smaller data structures would have likely been more efficient in practice.

The search function was implemented using regular expressions, specifically two different types. As documented above in the usage examples section, the user can choose the default partial searching or an exact searching method. The program applies these regular expressions to the code entry titles before checking the code snippets themselves. A list comprehension was used to find the title of a code snippet that contained a match to the user's search criteria.

Any further inquisition regarding the coding implementation should be directed to Nick Schappler at nschappl@nd.edu.

6. Discussion and Future Work

Overall, I'm extremely pleased with the results of my code library, especially considering it was only my second GUI design. I could see the structure of this library being used for a wide variety of subjects. For example, it would be relatively effortless to change this library for a foreign language major who wishes to maintain a library of vocabulary for a number of languages he is studying. During development, I became extremely familiar with the PyQt reference and will carry this information with me in my future coding endeavors. I know that I will use this code library in the future, and expect it to be a valuable reference for me moving forward.

Although I'm happy with my efforts, there are a few things I would like to improve in the future. I wasn't able to reach some of these 'checklist' items because of time constraints while working independently on this project. First and foremost, I would make titles editable like code snippets currently are. You would just need to verify that for the selected language, the revised title didn't already exist because the titles are used as unique keys in the data structure. I would create a more dynamic search that could still return results even if some of the words didn't match (much like a google web search). Also regarding the search, I would change what data is actually being searched or perhaps how code snippets are stored. Since code snippets are stored
as HTML, searching for an HTML tag would return results when you wouldn't necessarily want it to. For example, searching '<html>' would make all code entries match since they are all stored as HTML code. This would be unexpected behavior for the user. I would also consider the possibility of supporting multiple selection of languages. This would primarily be used for viewing and searching multiple language simultaneously. I thought about this possibility during development, but decided that in the short term, single-selection was a more streamlined approach for the user. I would also change how data is stored once the program is exited. Rather then using a hidden text file as I am currently, I would opt for the python module that relies on the operating system much more for these operations. This would be a safer and more reliable way to store and load data in between program uses. The last potential update would be to support more than just rich text as the code snippet. It could potentially be a nice improvement for a user to optionally store an image, per say, that could help him implement a certain algorithm.

7. References

PyQt Reference:
http://www.riverbankcomputing.co.uk/static/Docs/PyQt4/html/classes.html

Regular Expression Re Module:
http://www.regular-expressions.info/python.html

8. Author

An entrepreneur at heart, Nick Schappler is seeking a technical background in computer science as a junior at the University of Notre Dame. He enjoys playing or watching any and all sports, listening to music, and debating stock market plays with his friends. Nick is currently developing a web-based company at www.zyrow.com with two of his friends and looks forward to a cyber security internship at Booz-Allen-Hamilton in Washington D.C. this summer.