Opus PHP Editor

By

James Sauvé

Submitted to
the Faculty of the Information Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

University of Cincinnati
College of Education, Criminal Justice and Human Services

June 2012

Opus PHP Editor

by

James Sauvé

Submitted to
the Faculty of the Information Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

James Sauvé	Date
Russell McMahon, Faculty Advisor	Date

Table of Contents

Table of Contents iii

Acknowledgements iv

List of Figures v

Abstract vi

1. Project Description and Intended Use 1

1.1 Problem Statement 1

1.2 Solution Statement 1

1.3 User Profile 2

2. Design Protocols 3

2.1 Use Case Diagram 4

2.2 Class Diagram 5

2.3 User Interface 7

2.3.1 Interface Design/Navigation 7

2.3.2 Icons/Graphical Symbol 7

2.3.3 Color Scheme 7

2.3.4 Help 8

2.3.5 Code Clip Editor 9

3. Deliverables 9

4. Project Planning 9

5. Proof of Design 11

6. Testing 11

7. Conclusion 13

8. References 14

9. Appendix 1 – Acknowledging Sources 15

10. Appendix 2 – Difficulties Encountered in Development 16

Acknowledgements

I need to thank first and foremost my family. Without their support and encouragement going back to school and finishing a degree after so many years would not have been possible. I also need to thank my professors who have encouraged me and offered advice over the years. They are too numerous to list and span my time at various colleges in Virginia and Ohio. But I should mention by name at least Profs. Mark Stockman, Russ McMahon and Dr. Hazem Said. They have all been very patient and willing to offer advice and direction.

List of Figures

Figure 1. Use Case Diagram…….4

Figure 2. Class Diagram……...5

Figure 3. Main User Interface……..6

Figure 4. Code Clip Editor……...8

Figure 5. Gantt Chart – High Level……10

Figure 6. Project Expense Chart……10

Abstract

Opus is the answer to any developer's wish who wants a simple text editor for creating PHP web pages. This developer wants more than TextEdit or Notepad have to offer, but not something so heavy and over-featured as a full-blown IDE[1]. Opus provides a practical approach to creating PHP scripts, balancing the desire for features with the need to be agile. Of course, basic editor functions are provided, but also code formatting, keyword highlighting, launching files in your Apache document root folder, and more.

Opus is written in Java 6 so it can be used on multiple platforms though the principle development target here was Macintosh OS X (Lion). The footprint is very lightweight. The application itself is only about 3 megabytes with a memory load of 50 to 65 megs. This absolutely pales in comparison to Eclipse or Visual Studio, making Opus very attractive for any Java-enabled computer that may not be robust enough for those big IDE's.

[1] IDE stands for Integrated Development Environment, a comprehensive tool used by programmers for creating software. Common examples are Eclipse, NetBeans and Microsoft Visual Studio.

1. Project Description and Intended Use

1.1 Problem Statement

The internet and its related technologies have had an incredible effect on society.

PHP[2] is among the many internet programming languages that have emerged in the last twenty years, first created by Rasmus Lerdorf in the 1990's. It is a scripting language used for creating dynamic web pages. PHP is an interpreted language like JavaScript, but it runs on a web server, not in a browser. Other innovations include the numerous text editors used by developers to write code with. But it's here where the problem lies. Editors such as Eclipse, NetBeans and Komodo do a fine job. They also provide a glut of functionality and multi-language support. There's very little in the way of a text editor that is both very simple and dedicated to working with PHP.

What options are there for a simple, bare bones editor to work on PHP scripts with? Some possibilities include Notepad, TextEdit and gedit from Windows, Macintosh and Linux, respectively. These do work and some developers actually use them. But they are all general purpose editors, offering no specific support for PHP. Suppose a developer desires a text editor that supports PHP language syntax and file handling but also has a very light footprint on a computer, so it can run easily on a low-power device. Features to aid in editing and testing files would be nice too. In short, there must be a better balance between Eclipse at the one extreme and Notepad on the other.

1.2 Solution Statement

This is the problem Opus solves. Opus is an modest text editor that allows for the creation of PHP files as standalone files or as parts of larger projects.

---

[2] PHP originally meant "Personal Home Pages", generally it is understood now as "PHP: Hypertext Preprocessor", a recursive acronym.

With Opus, the user starts the application and faces an empty work container with a small arrangement of buttons and menus. Since this is a PHP-dedicated tool, there is no need to delve into a menu tree just to select a language or project type to work with. The user may begin a new PHP file just by clicking “New”, which will generate a new unsaved file containing boilerplate text to get work going. Or the user may click “Open” to locate and edit an existing file. There is also a menu item to quick-select recently edited files.

The developer enters what HTML and PHP they desire. Opus recognizes PHP reserve words and constructs, highlighting them in the text editor. Also, code can be “formatted”, that is, arranged in a more readable manner by clicking “Format”.

When the developer is finished, he or she can click “Save” to persist the file to their local system, even into the document folder of their Apache installation so they can run the file. Opus has the means to launch a PHP file from Apache using the local default web browser.

1.3 User Profile

The ideal user for Opus is any PHP programmer who would like a very simple, light tool to write PHP code with. Such an editor should be unencumbered by the excess of options and language support that one finds in larger tools. In fact, the prime candidate for Opus is someone who is seriously considering using Notepad, but really wants something “just a little bit more”.

Since this is a web programming tool, it is unlikely to have much appeal to the general public. But among programmers, it may be used by people having a variety of expertise levels. Novice coders may find the uncluttered interface a welcome sight.

Certainly, the highly focused purpose of Opus means a much lower learning curve than with a full-featured editor. There really isn't that much to learn. So most of the novice coders learning effort will be directed where it belongs, learning the PHP language. Opus even provides access in the user interface to PHP's list of built-in programming functions to act as a handy reference.

Even more experienced PHP coders can find pleasure in using Opus. The very simplicity of the application, and the lightweight footprint makes Opus easy to use on small devices that don't have enough power to handle a hungry full-featured editor. Opus is great for quick edits or creating new files.

2. Design Protocols

Given that the primary motivation behind Opus was to provide a text editor dedicated to PHP web page authoring, the design of the application had to follow suit. Thus, a simple user interface was created. For example, early iterations included buttons with icons. These icons were visually attractive and served well to suggest each buttons functionality. But they were removed in the interests of keeping to the theme of simplicity and securing as much screen "real estate" as possible. Overall functionality has been kept to those things that seemed most helpful for the user's workflow while avoiding an over encumbrance of controls. The buttons themselves are restricted to those functions that really benefit the user from easy access. Other functions may be found in the menu bar.

There is also a secondary "text editor" for creating code clips that a user may want to keep archived for periodic reuse. But even this was kept to a minimum of clutter. Only those buttons that seemed necessary to use the clips are given. The clip editor itself can be closed when not needed. In general, there are no controls consuming

space that don't need to be there.

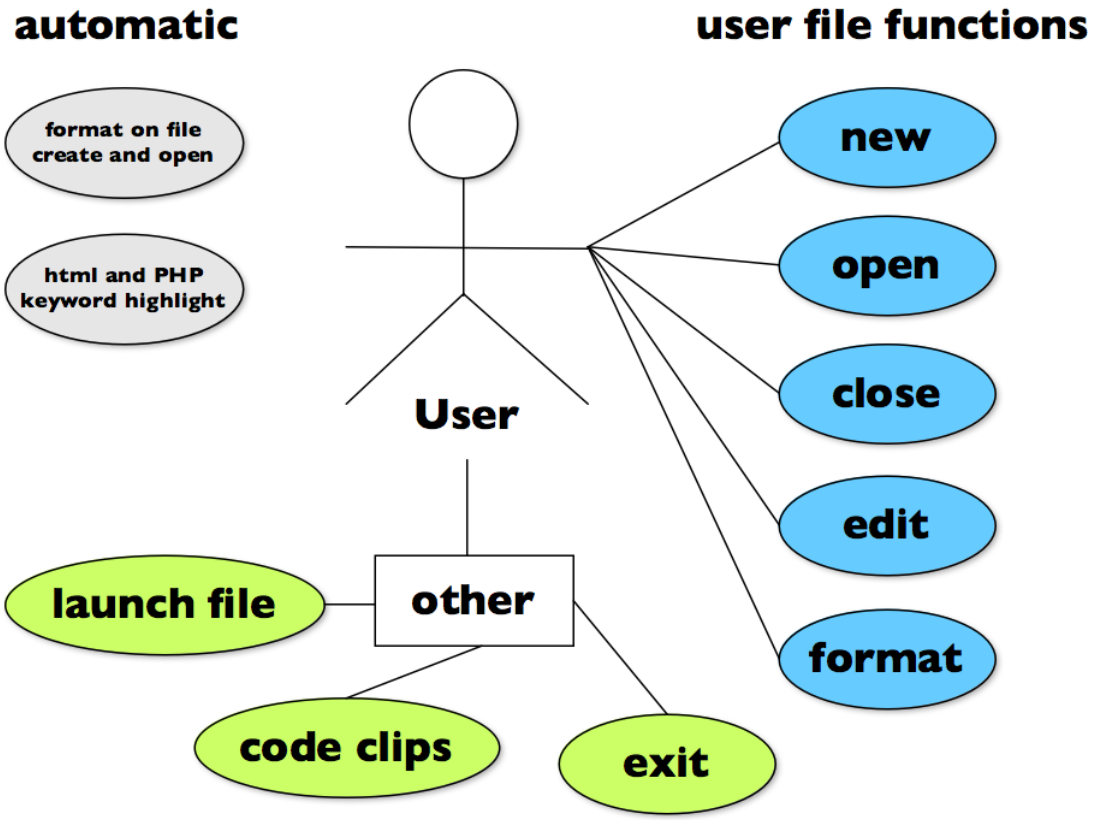

Figure 1. Use Case Diagram

2.1 Use Case Diagram

The Use Case Diagram is indicative of the overall nature of the project, strive for simplicity. There are no "users" versus "administrators", just a single "user". Other popular editors for creating code files do not make this distinction. It seemed superfluous to introduce the division in Opus as being inconsistent with the design theme. The basic editor functions are provided, as are aid in code formatting, highlighting, recent files used, code clips (mentioned above), and launching files from the Apache document root in the users default browser. This doesn't mean there isn't much going on "under the hood" of the application. But such activity is transparent to the user, whose attention should be focused on creating web content, not having to "fiddle" with the interface.

Figure 2. Class Diagram

2.2 Class Diagram

The system of classes represents a separation of concerns in the design of Opus. Various classes are dedicated to instantiated non-GUI objects, GUI objects the user will interact with, a group of classes offering operations to run on string text and files, and the Main class where the application is launched from.

Opus uses xml files to maintain information between sessions. This information includes the screen location and size of the application window, a recently used documents list, Apache root document folder path and subfolders for a) code clip text files, and b) PHP API files downloaded from php.net. The rationale for using xml files instead of a regular database tool such as MySQL was to keep the application installation simple. Using Opus depends on the presence of Java 6 and nothing more. Opus will

search for a hidden folder upon loading. The first time it does this on a given machine, obviously the folder will not be found. The files and folders will then be created with default content. The information in the files will then be updated over time as needed. Thereafter, when launching Opus, the data will be read and the application loaded with the saved values.

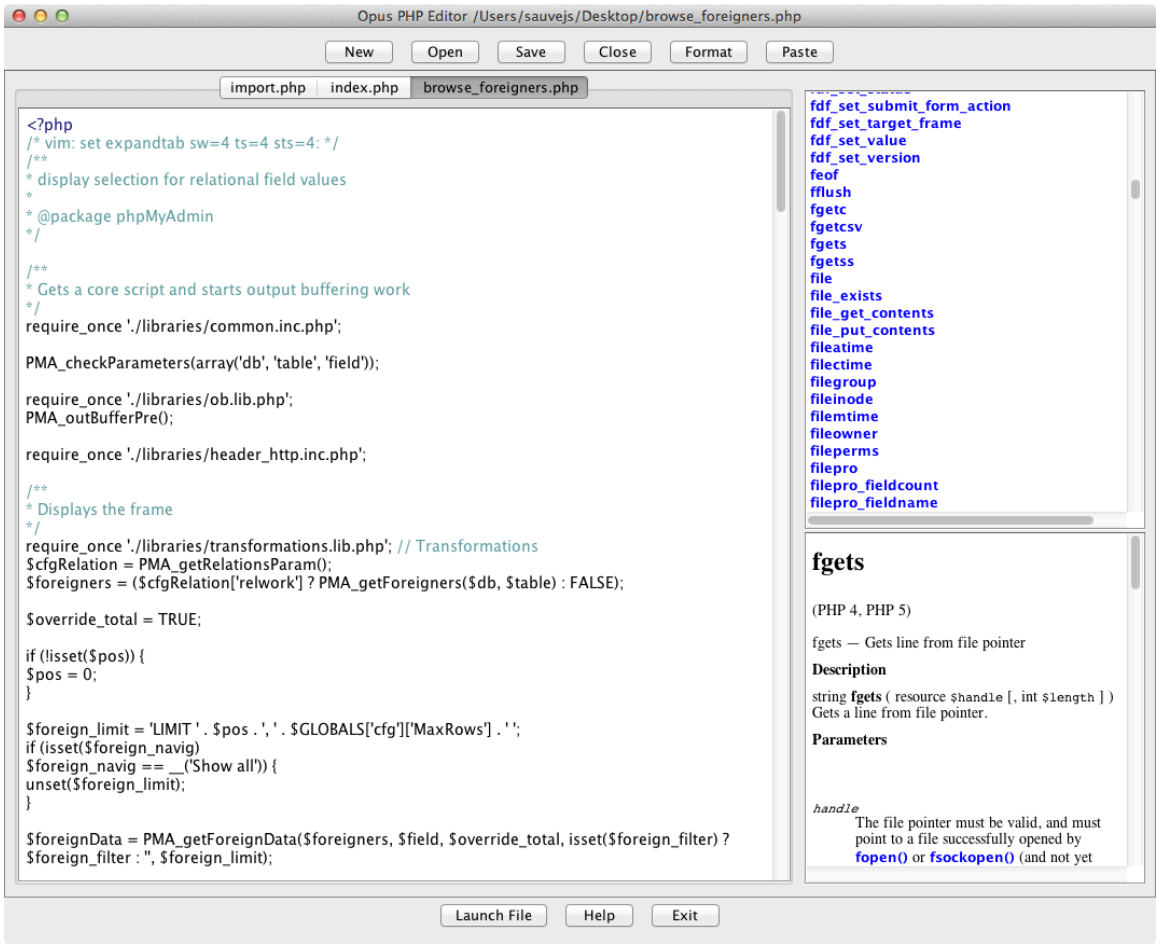

Figure 3. Main User Interface

2.3 User Interface

2.3.1 Interface Design/Navigation

Here, as noted, the design keynote was “keep it simple”. There should be no controls on the GUI that are not essential (admittedly, this is a subjective decision). There should be no decorations, no icons, no components that are not contributing to the basic functionality of Opus. Any useful, but not fundamental, operations will still be accessible from the menu bar.

2.3.2 Icons/Graphical Symbol

As noted, after initial trials with icon based buttons, the decision was made to remove them all, except for the one that resides in the dock on Mac OS X. This was done to preserve as much screen space as possible for work. Thus, the GUI of Opus is overall very clean and lacking in extraneous decoration. A person can argue this makes for a sterile looking program, but for smaller display screens the space efficiency justifies the bare look.

2.3.3 Color Scheme

Since this is a Java 6 based application, it is possible to run Opus on Microsoft Windows and various flavors of Linux. The primary development platform has been Macintosh OS X (Lion). It is possible to design a Java program to adopt the host platform “Look and Feel” or to use the default L&F provided by Java, called “Metal”. I have endeavored to make Opus appear as much as possible like a genuine Macintosh Cocoa application. Interestingly, some minor cosmetic issues arose when running builds for Windows or Linux. This required tweaking the GUI code depending on what platform was present. It also highlights how

Java's "write once, run anywhere" promise is not without some shortcomings. In the end, since I was targeting principally Macintosh and the flaws were negligible, I have not given them high priority in the debug process.

2.3.4 Help

The in-system help has yet to be developed[3]. Since one of the driving design protocols behind Opus has been simplicity and maintaining a low learning curve, creating a help tool will consist of only pointing out the functions of the buttons and how to launch dialog boxes such as Code Clips and the Apache root panel. Every effort has been made to keep Opus intuitive. If the application gets so complicated that one needs a book to figure it out, then by definition the point of the whole project has been defeated.

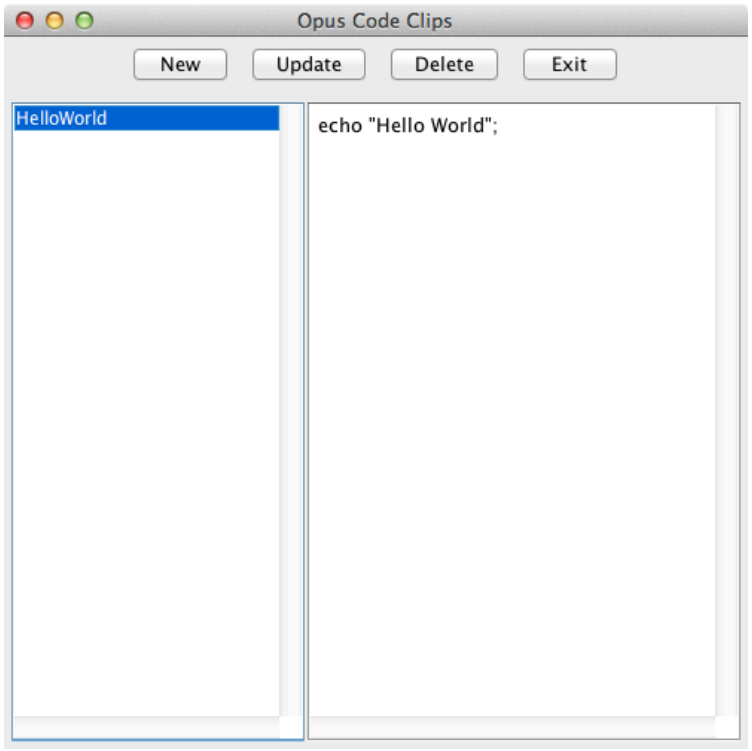

Figure 4. Code Clip Editor

[3] Later in this document, I will discuss some of the technical challenges and delays experienced in building Opus.

2.3.5 Code Clip Editor

The code clip editor's purpose is to allow the user to store "favorite" or oft-used bits of code for frequent reuse. The most obvious example would be connection strings to work with databases. But any sort of code block that the user finds handy to have always on hand my be stored permanently with this tool. Clips can also be edited as needed by making changes and clicking "Update" or simply deleted.

3. Deliverables

The project is now completed, what has been delivered is an application that fulfills the promise of the solution proposed above. The product provides the following:

- a text editor with standard edit functions (cut, copy, paste)
- automatic and on-call format and color code of text
- recent used files list
- archived code clips
- track open files with unsaved changes
- launch web pages in Apache root folder in default browser

4. Project Planning

The project has been completed in three stages in accordance with the senior design sequence. These stages included project formulation, prototype construction and final presentation.

The application was completed on time. Technical challenges precluded some accessory functionality from being implemented as originally desired, though the basic functions for a PHP editor are all present.

	Task Mode	Task Name	Duration	Start	Finish	Notes
		Concept Development	54 days	Wed 9/21/11	Sun 12/4/11	Fall '11 quarter, determine project focus and scope in consultation with faculty.
		Prototype Development	50 days	Tue 1/3/12	Sun 3/11/12	Winter '12 quarter, build prototype of project to demonstrate proof of concept viability.
		Project Completion	50 days	Mon 3/26/12	Fri 6/1/12	Spring '12 quarter, finish working model of project, demonstrate at 2012 Tech Expo and final presentation.

Figure 5. Gantt Chart – High Level

The above Gantt Chart gives a high overview of the three stages of the project development, from concept to prototype to completion.

	Projected Cost	Actual Cost
Hardware		
Apple iMac 21.5"	1800	0
principle development platform		
Acer Aspire 6850 laptop	400	0
used for inclass demo using Linux		
Software		
Eclipse Java JDK	0	0
development IDE		
VMWare Fusion 4	50	0
allows running Windows or Linux on a Mac		
Ubuntu Linux	0	0
alternate target platform		
Microsoft Windows 7	10	0
alternate target platform		
MAMP	0	0
(Apache/MySQL stack for Mac)		
Microsoft Office 2011 for Mac	0	0
student edition, UC Bookstore		
Shapes	5	0
alternative to MS Visio, Apple App Store		
Total Cost	$2,265	$0

Figure 6. Project Expense Chart

The expense chart above estimates the cost of hardware and software for the project if all materials had to be purchased for the creation of Opus. As it is, the two computers and any proprietary software were already purchased by the author beforehand. Other software, such as Eclipse and Java, are free for anyone to download.

5. Proof of Design

The application is now complete at the essential level. It can be used for its final, intended purpose. This means the application can already be used as a text editor, providing all basic edit functions, file functions, code highlight, indent help and file launch.

The principle design criteria was simplicity. To learn a programming language requires time and effort. A full-featured IDE can distract from that with too much complexity. To the extent that it was the intention to keep this theme, simplicity, then the application is a success. Opus provides a user with a clean, elegant means to write PHP code with features that are expected and useful without accumulating pointless cruft. Files can be created, edited, saved and recalled, launched from your default browser, and language API assistance is provided.

6. Testing

Opus has been tested for a variety of functions including the following.

Essential Text Edit

This includes copy, cut and paste text content, to and from outside the application.

Recent State Data

Opus will keep track of recently used files and recall, when starting, the last location and size on screen from the previous session. The path of the local Apache root folder is also maintained.

Visual Status Indicators

The application will indicate to the user when there are unsaved changes to a document and display alert pop-ups with helpful messages where useful (i.e., selecting a file from the recent files list that is no longer available, closing with unsaved changes, etc.).

Keyword Highlight

Opus will display most text in black font. HTML and PHP reserved words, and comments, will show in color to easily distinguish them for the user. This feature has been the source of the most problems during development and will be discussed in detail below.

Content Format and Indent

While highlighting keywords and comments in color, Opus will also arrange code in practical spacing and indentation for ease of reading.

Insert Code Snippets

This allows the user to inject saved pieces of pre-defined code into a document to speed up work.

All functionality of Opus has been tested and found to work acceptably as described here with the exception noted that will be discussed later.

7. Conclusion

The purpose in creating Opus was to answer what I found to be a gap in the field of available tools for authoring PHP web pages. Looking for an editor that was at once useful but also simple enough to learn quickly, I found what was out there generally either too complicated and laden with too much functionality or so bare-boned as to be insufficient. I wanted something that struck a balance between the two extremes represented by large IDE's like Eclipse or Visual Studio and plain editors like Notepad.

Opus fulfills this need by offering a platform for creating PHP based content that is easy to use and helpful. Opus will do this by offering enough features to be helpful, make for an easy workflow and simple environment. It will be a symmetry of usefulness and minimalism.

8. References

Eclipse, the editor I will use for the project:
http://www.eclipse.org/

The Java API for version 6, the minimum version needed to run Opus:
http://docs.oracle.com/javase/6/docs/api/

PHP's main site, this is the web technology I will be building the tool for:
http://www.php.net/

The distro of Linux I will use for testing Opus as well as Mac OS X:
http://www.ubuntu.com/

The 3WC's validator service, to verify compliance of the files my tool creates:
http://validator.w3.org/

Safari books online via UC libraries:
https://login.proxy.libraries.uc.edu/

The search engine used to find answers on the web:
http://www.google.com/

Stack Overflow, the best help forum site I know of
(this was used more than any other code help site):
http://www.stackoverflow.com/

A couple other sites with examples:
http://www.coderanch.com/

http://www.java2s.com/

Wikipedia, general online information:
http://en.wikipedia.org/wiki/main_page

9. Appendix 1 – Acknowledging Sources

As noted above in the references, some web sites have been very helpful in providing guidance for solving problems. On commencing this project I was completely unfamiliar with building a Swing interface. My Java was, honestly, fledgling. I still have a lot to learn. But these sites have been a great help in giving me aid. There are other sites that provided varying amounts of assistance, but the ones noted above are by far the ones I have availed myself of the most.

The code is, therefore, generally mine except to the extent I found answers for difficulties by searching online, and usually finding what I needed on StackOverflow. There are also some third party Java libraries I have employed to assist in creating Opus.

They are:

1. **AppleJavaExtensions.jar**, this is a library used for a variety of things, for me it facilitated making a Java application on Mac OS X that would look just like a real Cocoa app that had been written for the Mac. You can get it here: https://developer.apple.com/library/mac/#samplecode/AppleJavaExtensions/Introduction/Intro.html

2. **guava-11.0.1.jar**, a library from Google. I used it to read files, which it did handily, though it does a lot more. Get it here: http://code.google.com/p/guava-libraries/

3. **jdom-1.1.2.jar**, another library I used to read files, namely xml files. It can be downloaded here: http://www.jdom.org/dist/binary/

4. **jsoup-1.6.1.jar,** a library for parsing HTML to extract sections of pages in HTTP responses http://jsoup.org/

5. **stringsearch-2.jar,** a library for quickly parsing through a string to search for matching groups of characters http://johannburkard.de/software/stringsearch/

It was never my intention to simply "rip off" other people's code, without proper credit being given. Thus the acknowledgements and links provided here. Moreover, without the Apple jar file, the convincing simulation of the Cocoa "look and feel" would simply not have been possible. So the use of it was unavoidable, unless I wanted to have a "look and feel" on the Mac that really wasn't genuine.

The other .jar files were used partly to try out some open-source projects and also to avoid reinventing the wheel. The last file was tried in my attempt to speed up the process of parsing files to set keyword highlights.

10. Appendix 2 – Difficulties Encountered in Development

As I have noted previously in this report, there have been difficulties encountered while creating the application, Opus. Problems are not uncommon is the process of building software. But in this case, there was one particular issue that proved insurmountable.

Opus is basically a text editor, an editor I had built with coding PHP web pages in mind. One of the common behaviors of such editors is to highlight those keywords, sometimes called reserve words, that are special to a language. Since PHP pages typically include at least some HTML tags and not just the two PHP tags, "<?php" and "?>", these need to be looked for as well.

So it became a goal to build an algorithm that would look for such keywords each time the user struck a key on the keyboard, thus attempting to keep all highlighting up-to-date. As soon as I finished typing "<html" then the algorithm would match this string to the appropriate styling I had defined for HTML tags. It would then find the matching ">"

and style it in the same manner. Anything in between, such as inline CSS or a bit of javascript, would be unstyled. The algorithm would do likewise for comments and PHP tags, only the style defined for them was different.

Initially, I was quite pleased with how well I had gotten this to work. I had a mechanism step through the text in the given file, character by character, trying to match the beginning of the text with anything in my List<String> of tag names. If a match was found, the substring would be styled appropriately and appended to a Document object. If no match, then the single character would be appended unstyled and the process would start over with next character in the text.

Again, this worked really well. It worked, that is, until I started more rigorous testing. I found much to my dismay that large files were very slow to process in this manner. In fact, they were very slow just to load into Opus. The styling process happens automatically every time you open file. I tested some large files I copied from the phpmyadmin folder on my machine and found that files with upwards of 10,000 characters were basically useless. The algorithm was just too slow grinding through so many characters every time I hit a key. I could actually see the styling effect getting applied when matches were found. It was like watching phosphorous glow.

I needed a better solution. I tried several rewrites and kept coming back to basically the same problem. Finally, I found online a jar file that did offer some help, stringsearch-2.jar. I also got some ideas from browsing the web. The jar file looks for the first instance of a string in another string, supposedly faster than the method built into Java's String class. I built a method to go through all the tags just looking for which tag was closest to the "left" of the string (i.e., which had the lowest index). Now I would just

do a search of each tag and keep track of which had the lowest index. If a match was 0 index, just append that to the Document object with appropriate styling. If the lowest index was, say 10, then append the substring 0 to 9 unstyled then append the tag with correct styling. If no match, append what ever was there unstyled.

Upon testing this new method, I was delighted to see it was much faster than the old version. Large files now loaded acceptably fast. Unfortunately, they still didn't work acceptably fast. It was an improvement, no question. But I really wanted this part of Opus to work well. It does work fine for small to medium files (granted, that's a subjective description). If the character count is less than a thousand, it should be fine. But that is not what I had been hoping for. I wanted this tool to work as well as Eclipse or Visual Studio. No matter what my tweaking and playing around, I simply could not get the syntax highlighter to zip through all the content of a large file fast enough. With large files, typing becomes impossible. You're constantly waiting for the algorithm to catch up.

I did manage to find on StackOverflow a brief high-level description of what the folks at Eclipse do to accomplish this. Apparently I am not the only one out there who has tried building such an editor only to be stymied by this problem. See the discussion here:

http://stackoverflow.com/questions/7243409/syntax-highlighting-how-does-eclipse-do-it-so-fast

I found this discussion shortly before the end of the third quarter. It was at this point I essentially decided this was a technical issue that:

a. maybe I could solve, but I don't have enough knowledge of Java yet
b. I just don't have the computer science or math background, too bad
c. there might be limitations in the Java Document object and the JTextPane I was using, so it's not all my fault

I suspect it's a combination of a and c. I don't think this really takes a genius to figure out. I do have a lot to learn about Java. But it's also possible the basic Java tools are simply not up to the job. The discussion on StackOverflow actually doesn't say Eclipse uses such tools. Maybe they created their own just for this purpose, or made extensions of the Java classes. I don't know. I have seen others posting online complaints about how slow JTextPane is.

Also, the Document object can only append or insert strings that are styled in one style at a time. In other words, it would have been nice to be able to take all the text from index 0 to just before my cursor and just copy it "as is" into a Document object. Then do the same for after my cursor to the end. Then the algorithm would only have to parse a tiny subset of the whole document. But Java won't allow that.

See the insertString method here:

http://docs.oracle.com/javase/6/docs/api/javax/swing/text/Document.html

There is no overload. Any styled text must be inserted with one style per insert. Thus, multistyled text required multiple inserts. I may experiment with seeing if I can detect beforehand what substrings are styled with what style and use that to reduce the inserts. That may work.

At this point though, my regret is that the program does function but that it suffers from a performance issue that I haven't been able to overcome.