

Bible Reader

ver 1.0

By: Karim Abdul

Final Project - CSCI 4237/188

12/10/2010

Introduction

A bible reader application was written in Java Micro Edition (J2ME) SDK to accomplish a sound prove concept of readable bible for the mobile system. With the limitations of memory, library availability and speed; the project itself seemed quite bit challenging over all. The project encompasses two java file as rhMIDlet.java and customCanvas.java. rhMIDlet file does the entire parsing, populates the list of the Book on the form followed by the list of Chapters within the inner layer of the Book; whereas customCanvas file does the text wrapping and displays the scroll bar with respect to text in a dynamic fashion. The contrast of this paper would give an overview of what these both files does, and how do they function dynamically to achieve the end result.

Background

As we are all aware what Bible is, however it is important to be acknowledged with few important criteria as of where does the challenging aspects come in place. Bible tends to have different books, followed with different chapters, followed with different verses. This three-tier level of abstraction method is what gets the entire level of parsing into difficulty level, and thus some logical approaches are necessary to take, to achieve satisfactory results. I used King James Version of the Bible for this project respectfully.

Start-up

Once the program have started, rhMIDlet get initiated as a main MIDlet, and triggers a alert function to check the capacity of the memory that is required to execute the entire application. If the required memory is less than the actual memory, then the program just exits, or else jumps to parsing functionality. While the program is parsing, the screen indicates the program is actually loading. The loading functionality takes few seconds, in which the entire Bible gets downloaded, appended and parsed accordingly.

The program requires minimum 10mb of space to complete its entire parsing and loading functionality, thus this is one of the constraints for this project and had to take extreme measures to understand where the most memory is being leaked, and how can I clear few of the un-used objects after their task have been accomplished.

At the very beginning of the parsing, the program downloads the file of Bible from http stream, and stores the content into the memory. Via the http input stream, we can get the entire length of the Bible, therefore that value stored as length give us the entire range of the offset values for the book. So for example if we read the entire length of the Book as 4921540, we can certainly tell by that value the Book is 4.9mb in size, and it has 4921540 characters all together. This satisfies one of our conditions by giving us the range as to where to end until, certainly 0 is

our first character offset value and the file length is the last, which we didn't know before archiving the file content.

Data Structure Mechanism

The parsing mechanism I have conducted is fairly easy; however the data structure of the entire system must be known by actually looking at the book and sensing as of how the books might need to parse accordingly. A fairly good example of parsing method which I used in this project was that every new Book had a sentence starting with Book, with its numerical representation and then the name of the actual book. For example if I am trying to allocate "the Book Ruth", then I would be interested in getting the **integer offset** values of "Book 8 Ruth", where Ruth in Kings Bible version tends to be the 8th Book. By gradually understanding the data structure behavior and sensing what values could be utilized to get the parsing done accordingly, the implementation of the parsing gets easily done without any major issues. Therefore, the first primarily task I did for my first-level parsing was to get entire integer offset values of "Book integer", where integer officially starts from 1 and ends until 66. Where the number 66 does comes from? The total number of Book's written in Old and New Testament in King Bible version accumulates to total 66 numbers of Books in total, so therefore I know by that what the cutoff limit is. The Book integer offset value is thus stored into a vector.

Once the vectors have been stored for the Books, iteration occurs for Chapters with respect to every book integer offset value captured. So let's for example I am trying to find the Chapters in book 1, the range I utilize is from Book1 offset value to Book2 offset value. Therefore from this method I capture the chapters offset values of each book and store it in vectors form. The main particular question comes in place as of how do I interact each book versus its chapter? For such abstract a creation of "Objects" is necessary. Each book object consists of the Book value versus the chapter vectors reference. So by this I am able to get hold of my Chapter integer offset values as of what is the range of integer offset values of each chapter with respect to each book. The only major problem occurred for the last book as there is no extra book to set the end range, therefore I had set the file length size for the end range. Since we already know that there isn't any new Book listed after 66th, it makes it easier just to compare your last book range with the filesize offset values, to achieve the remaining chapters offset values.

Display

The text version of the display was being conducted in Canvas display, in which there is no automatic text wrap or scroll bar to correspond with the actual text. The approach of text wrap and scroll bar both were done dynamically by achieving the width and height of the screen, and by the width and height of each font character. By this we can easily accumulate how many lines can appear on the screen with respect to the entire text. Thus then the challenging aspect tends to

as of how to control the increment and decrement of each line versus the entire text, however before that let me emphasize as of how I emphasized text wrap approach. The text wrap approach was conducted fairly by first detecting the cutoff of the screen through the getWidth function. As the character reaches the width of the screen, it checks the condition of white space, and decrement its position of the character until white space is detected, once the whitespace is being detected within the character, new line is being apprehended, and the character position thus starts right after the whitespace position, which again tends to be the beginning of a new “word”. By this approach I could easily accomplish the x-plane positioning of the text wrap. However the height was bit more tricky part. To accomplish that, I had to get the size of the font. Once, I could accumulate the active Fonts width and height, I could calculate as of how much text could be shown on the active screen. By this calculation you could also set a counter for the number of lines **remaining** to be shown with respect to the total number of lines is **being** shown on the active screen, this gives the control for scroll bar to determine the use of active zone versus the accumulation of the text encountered as in whole.

The Scroll bar approach was merely being conducted as emphasized earlier by accumulating the text encountered in total, against the text shown on active screen. The text shown on active screen had a counter, which would roll back and forth against the total text lines count. So let’s per example, I have a text which fits 20 lines on screen and have 50 lines in total, therefore the range of count for active screen tends to be from 0 to 20 as 0 in count, for any addition line would increment that count, until the range reaches to 50. The same method would work the opposite manner if the scroll bar needs to be scrolled up. Once this was being accomplished, it was just the matter of apprehending the repaint method over canvas, to show the tendency of the active portion of screen versus the total length of lines.

Chapter Flip Flop

There is a control mechanism of chapters within the Midlet of the canvas for flipping between chapters of each book. Since, the user might want to read a complete book, I thought it might be nice to have controls of flipping through each chapter within the Canvas, rather than going back to the list of chapters after completion of each chapter and allocate another chapter within the list. Since I already had the offset values of integers stored in vectors, the calling of previous/next chapter was easily accomplished by calling previous or next element within a vector. Thus again a count method could easily accomplish this job going back and forth in between the vector length.

Problems and Issues

The most challenging issue I ran into was utilizing the memory allocations to execute the entire program effectively. Due to the nature of involving http input stream content for downloading the file, the file itself took about 4.5mb in size to complete the download; from

there the parsing of file method took another 5mb for processing. This heavy utilization of memory required some tweaking of the profile for a particular emulator of the phone, in which the heap memory size needed to be increased gradually. Although JavaFX, one of the emulator in the list does seams to provide total 20mb of the free memory allocation, but it is also important to know how to tweak the profile of the emulators if we fall into certain condition of having no other choice. To allocate free memory, I had to do some intensive reading, in which I did happen to find a solution of utilizing Runtime garbage collector, to achieve free memory when declared empty object or null parameter after utilization. I had to flush the memory content once I had utilized the objects such as “stringbuffer”, “vectors” or “close http stream” to flush memory for optimization purpose.

Another issue I came across in this project was utilizing the encoding method of which the file supported. I had major problems using ANSI encoding as the content of text after some lines wouldn't show properly. I came across this solution by changing the text encoding method to UTF-8, and it fixed this problem. It is also a good set of practice to set your http properties as the type of encoding you will use throughout the transaction, therefore you maintain the standards of the encoding throughout the parsing and display level.

One common issue for such project tends to be for checking the boundary conditions of the books and keeping it synchronized with the self contents of the chapters representing by counts within your system. The count in for loop and vectors generally start from 0 not 1, and the book value count started from 1, same for the chapter, keeping such boundary conditions in mind, I had to keep a thumb rule for every scenario as $(n+1)$, with respect to when reaching the boundary to check the condition $(length()-1)$. Keeping in mind of such subsets of conditions worked throughout the course period of project, in which the results could be obtained deliberately in form dynamic fashion and gave less to none errors.

Conclusion

I am pleased with the overall satisfactory result of my project, and satisfied with the working results as I had planned for during the proposal period. My main concerns were keeping most of the form of this project into dynamic state, and keep it simple as possible. During the process of the project development, I came across different sets of challenges and problems, however, such challenges were important as it much helped in understanding the entire flow of the design, data structures, algorithm foundation, storage and processing regulations and overall using the system under entirely strict sets of libraries.

The program is overall a classic version of Bible Reader we see in online market today. It is easy to use and overall it works very well within the J2ME platform.