Western Oregon University
Digital Commons@WOU

Honors Senior Theses/Projects | Student Scholarship

5-1-2015

Painting With Data

Tristan Knope-Jenkins
Western Oregon University, Tknopejenkins10@mail.wou.edu

Follow this and additional works at: https://digitalcommons.wou.edu/honors_theses

Part of the Computer Sciences Commons

Recommended Citation

Knope-Jenkins, Tristan, "Painting With Data" (2015). *Honors Senior Theses/Projects*. 51.
https://digitalcommons.wou.edu/honors_theses/51

This Undergraduate Honors Thesis/Project is brought to you for free and open access by the Student Scholarship at Digital Commons@WOU. It has been accepted for inclusion in Honors Senior Theses/Projects by an authorized administrator of Digital Commons@WOU. For more information, please contact digitalcommons@wou.edu.

Painting With Data

By

Tristan Knope-Jenkins

An Honors Thesis Submitted in Partial Fulfillment
of the Requirements for Graduation from the
Western Oregon University Honors Program

Dr. Becka Morgan
Thesis Advisor

Dr. Gavin Keulks,
Honors Program Director

Western Oregon University

May 2015

Table Of Contents

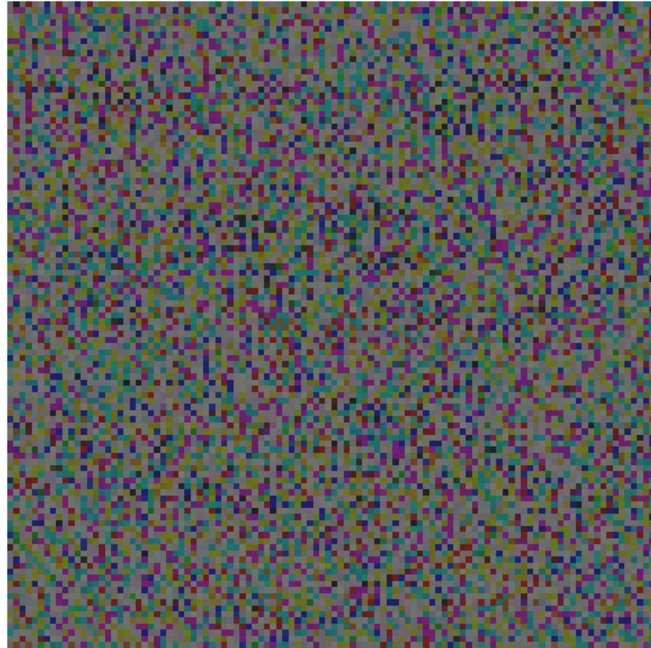
1. Abstract
2. Introduction
    - Background
    - Helpful Terms
3. A primer on Numbering Systems
    - Decimal
    - Binary
    - Hexdecimal
    - Encoding UTF-8 & ASCII
4. Research
    - File Formats
    - Metadata
    - Textfiles
    - Bmp/Dib
5. Project Specifications
6. Challenges
    - The Problem with Endians
    - No Null Characters
    - The ExtendedByte
7. Results
8. Acknowledgements
9. Bibliography
10. Appendix: The Code

Abstract

At the bit level there is no difference between a number, an image, a word, or a sound, they are all binary strings. I plan to find the parts that say that “this is a picture or this is a text file” and write a program that gives us the power to change how we experience that data. Depending on what my research turns up as to the way file types (mp3, jpg, txt) are encoded; I will either change the files at the byte level or create an arbitrary system to translate data into other mediums. Once this program is complete I will open source the project so that it is available for anyone’s use.

Introduction

Background



A number of years ago when I was taking an entry level computer science class, my professors posed a question. How does the computer know what to do with any specific file? There must be a sequence of bits somewhere in the file that tells it exactly what to do with it. I would think that if that's the case, there would be a way to change those bits so that the computer is tricked into opening a file as something other than it normally would.

That comment stuck in my mind as I took future computer science classes. As I advanced through the program and my skills grew, so did my curiosity as to what a picture would sound like, or what a book would look like. Later, I saw a short video of someone creating a C program for Hello world. But they didn't use a normal Integrated development environment to do so, instead they used Microsoft Paint. As they carefully chose each colored pixel I became even more curious as to how this fantastic trick had been accomplished.

So when it came time for me to decide what to research for my senior thesis the answer was simple. I would discover how it was that these tricks had been done, and

what The Hitchhikers Guide to the Galaxy (image above) would sound like, what Beethoven would look like, and what the secret to painting programs was.

During my research over the course of this year I was able to find the answers to some of these questions. I figured out what the trick to painting programs was. I developed a program that could turn a huge amount of text into an image, and I learned some valuable lessons about the Endians. To make this story meaningful there are some things that I need to explain first.

Terms that may be helpful:

String: A linear sequence of characters, numbers, or words.

Bit: A single zero or one binary digit.

Byte: A string of 8 bits.

Metadata: Information pertaining to a file that the user does not access directly, such as size, the date it was created, and various other details important to the computer.

Array: A type of list used to store data linearly.

Refactoring: The process of changing code structure or design in order to make it cleaner or more effective.

Offset: The number of bytes from the beginning of a file that a specific piece of information can be found.

Integer: A 32-bit signed two's complement (can hold values up to 31-bits in length) number.

A Primer on Numbering Systems

Decimal

We use numbers every day of our lives, and the type of numbers most of us use are referred to as Decimal. This means that the numbers are based off of a Base-10 positioning system. We have used this system all our lives, so it shouldn't require any additional explanation.

Binary

Binary is a base-2 number system. So the only digits that exist in the system are zeros and ones. Every other number is made up of some combination of those digits. Each sequential position represents an additional power of two. To find the binary value of a decimal number you find the biggest power of 2 that is less than the decimal number, make columns equal to that power, jot down a 1 in the leftmost row, and subtract that number from your original value. If you repeat this process until the remainder is zero, that line of ones and zeros is the binary representation of the number you started with.

0	0	1	0	1	0	1	0
128	64	32	16	8	4	2	1

Computers use binary because the basis for transistors (a big part of computing technology) is that they can either be on or off, a one or a zero. All of our more advanced data storage evolved from this technology, so computers continue to use binary as its primary method of storing data. (Stallings. 2000)

Hexdecimal

So if decimal is Base-10 and binary is Base-2, then Hexadecimal (often shortened to Hex) is Base-16. Because our number system is primarily intended for using decimal numbers, we don't have numeric characters for the numbers 10 through 15 (well there's 10-15 those are 2 digits so writing "15" in hex would represent the decimal value 21). Instead we borrow from the alphabet and use 'A' through 'F' to represent the values of 10 through 15.

Since computers work in binary, and humans work in decimal, why talk about hex? You can probably spot the difference between, 01100111 and 01000111, but what about between 0110010100101010 and 0110001100101010? The difference in the first set is only 32, but in the second set the difference is nearly a thousand, and we are still only looking at half of the bits that a computer uses to store a single integer value. In hex however the two sequences above can be written as 652A and 632A, and spotting the difference becomes trivial.

Performing the conversion from binary to hex is also quite simple. Take the string from above, we can take it and break it into groups of 4 bits and then translate each of them individually.

Binary	0110	0101	0010	1010
Decimal	6	5	2	10
Hex	6	5	2	A

UTF-8 And ASCII

We have covered storing numeric values but text works a little differently. Since 2008, **U**niversal Coded Character Set + **T**ransformation **F**ormat - **8** Bit (UTF-8) has been the standard for storing text. Even though UTF-8 encodes thousands of characters, most of the codes we use everyday are based its predecessor: the American Standard Code for Information Interchange (ASCII). Each character A-Z (both capital and lowercase), digits 0-9, common punctuation/symbols, and a variety of formatting characters, 128 in total, are encoded as 8-bit binary sequences.

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
0	0	NUL	16	10	DLE	32	20	(space)	48	30	0
1	1	SOH	17	11	DC1	33	21	!	49	31	1
2	2	STX	18	12	DC2	34	22	"	50	32	2
3	3	ETX	19	13	DC3	35	23	#	51	33	3
4	4	EOT	20	14	DC4	36	24	$	52	34	4
5	5	ENQ	21	15	NAK	37	25	%	53	35	5
6	6	ACK	22	16	SYN	38	26	&	54	36	6
7	7	BEL	23	17	ETB	39	27	'	55	37	7
8	8	BS	24	18	CAN	40	28	(	56	38	8
9	9	TAB	25	19	EM	41	29	)	57	39	9
10	A	LF	26	1A	SUB	42	2A	*	58	3A	:
11	B	VT	27	1B	ESC	43	2B	+	59	3B	;
12	C	FF	28	1C	FS	44	2C	,	60	3C	<
13	D	CR	29	1D	GS	45	2D	-	61	3D	=
14	E	SO	30	1E	RS	46	2E	.	62	3E	>
15	F	SI	31	1F	US	47	2F	/	63	3F	?

ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

Research

File Formats

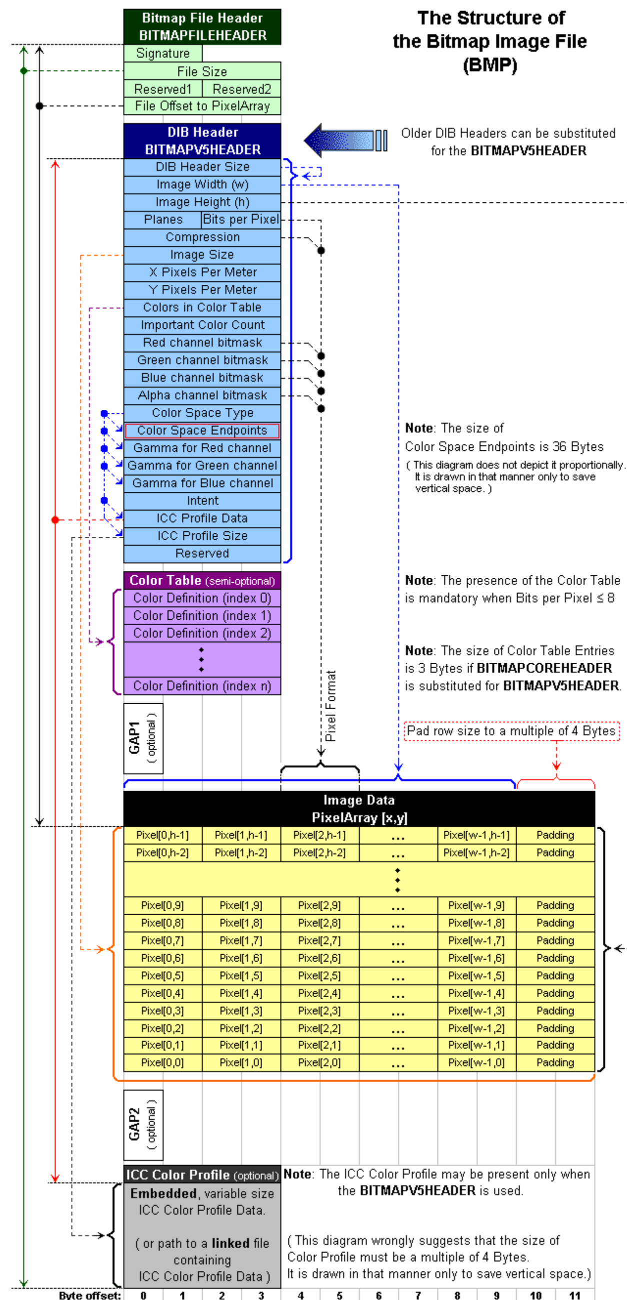
One of the intrinsic requirements of this project was that I build a strong familiarity with the way files are formatted and stored on computers. I spent weeks reading through the technical specifications of various file formats. Most people are probably familiar with file types such as zip, pdf, jpg, mp3, doc, odt, or rar; but there are more than a thousand different types (so many that wikipedia has to break up their alphabetical list into 4 different pages).

It was relatively early in my research that I discovered that if there was a single bit to twiddle and change how the computer interprets a file, it would be part of the files metadata. As I pursued this avenue of investigation I came to an unfortunate, albeit not entirely unexpected conclusion, the metadata for each file type is so unique that you cannot simply change a small number of bits and expect the computer to be able to interpret the result. The metadata says as much about how a file should look as the actual data component of the file.

Metadata

The term meta refers to an abstraction of a concept used to complete or add to the original concept; something that is characteristically self referential. A file's metadata is information that while used by the computer when it is reading or modifying a file, but it's not data that actually appears anywhere in the result of the computer's display of the file. Some metadata is information about how the file should be processed, in other cases

it refers to the file's size, copywrite information about the file, who owns it, or when it was last modified. Additionally many images have codes for what kind of camera was used, or other kinds of information that the creators of the file type considered to be relevant at the time of the files inception. The task in changing a file from one format to another, is to rewrite its metadata.



Text Files

So from here, the course of my research changed to an analysis of how files metadata was structured. I found that text files (txt) are the simplest file type. They hold no metadata whatsoever, and can therefore exist as a file with a size of zero. This trait makes them ideal for being a neutral point to start my conversion process. All that will be required to recreate them as images or sounds is to add the appropriate metadata, and because the ASCII encoding of a text

file has a direct correlation with binary, text files are a source of huge quantities of binary data.

Bmp/Dib

The next choice I had to make was what to turn my binary file into. Because we are so naturally accustomed to processing information visually, and because I believed that it would bring me closer to the secret of painting programs I chose to look into image formats that I could make the data fit into. After doing extensive research on image formats, I settled on working with the Bitmap image file (also called Device independent bitmap). I choose the Bmp/Dib file because it has relatively simple metadata and the information pertaining to the image itself is stored entirely in an array at the end of the file. The image above provides a detailed look into the metadata of a Bmp/Dib file.

Project Specifications

My goal was to create a program that would take a file of one type and turn it into something else entirely. It was also important that this project be able to effectively show how all files are built on an underlying structure of binary data. I choose to write this program in Java since I have the most experience with it. Java was also a logical choice because it's widely supported and has a robust selection of libraries built for it.

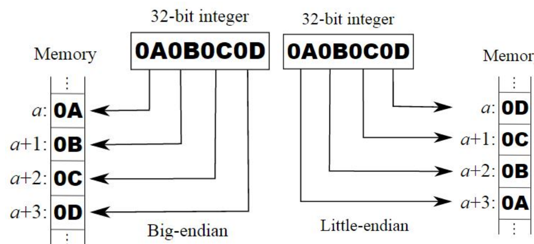
As I determined during my research the primary challenge will be correctly generating the metadata to turn a txt file into a bmp. In order to do this I will have to generate metadata that is appropriate to an arbitrary text input. I also believe that it is important that I create within my project the framework required to quickly and easily support new types of files for conversion. This means I will have to focus on creating generic data structures and building class interfaces that scale well as the size of the data or the size of the project itself grows.

Challenges

The Problem with Endians

With this project and with most things in life, it was the challenges that made it interesting. With each of the many challenges that this project brought, I was forced to learn something new to overcome it. The first major roadblock that I hit came while working on trying to process the metadata of an existing bitmap. My research showed that the offset for the size of the bitmap was two bytes, but when I translated the binary at that location I came up with an enormous negative filesize. Since I was fairly certain that the file I had passed in was not of some miraculous negative size, I started to look for the mistake.

I knew that the problem was an integer overflow error (when the leftmost bit is written as a 1 the value becomes negative). It wasn't until a few days later that an offhand remark by Dr. Bob Broeg about bit significance led me to the underlying cause. Bit significance ended up being one of the biggest (and most perpetual) struggles that I faced throughout the project in the form of endianness. Endianness is a convention used to interpret the data making up a binary string. If when data is stored, the most significant bytes are in the lowest memory addresses then it is referred to as being Big-endian, and if the least significant bytes are stored in the lowest addresses then the data is in Little-endian. When designing a new file format you can choose whichever endian you prefer, you will then need to stick with it in order to ensure backwards compatibility.



No Null Characters

Once I worked around the initial problems surrounding endianness, I started to write the algorithm to generate metadata for an arbitrary text file. There were numerous places in the specifications for bitmap that required empty bytes. However, when translating from strings to bytes, the built in methods doesn't take your input literally, so in the places that I had intended it to put the hex value ‘00’, instead it used ‘30’, the ASCII value for 0. Eventually I determined that the best way to solve this problem given my current setup was to use a “null character” since it has the ASCII value of ‘00’. Actually writting this character took a little bit of work though, since there is no keyboard button for null.

This solution to the problem was short lived however. While it did end up working on the small scale, when I tried to create an image larger than two pixels by two pixels this method caused issues in other parts of my code where numeric values were being translated into byte values. Eventually I came to realize that a more elegant solution than using strings to handle the creation of bytes was going to be necessary.

The ExtendedByte

This led to the Extended Byte, a major refactoring that occurred to improve the scalability and effectiveness of my current code. I choose to call it extended because it would be necessary for it to handle numbers larger than 2^8 (the limit for the byte data class). This class uses a linked list to hold an arbitrary number of bytes, it is able to make use of the Byte Buffer library method to translate an integer value into a byte array. From

here I use a Byte Literal initialized to zero (another of those little things I wish I had known about at the start of the project) to handle the places where there should be null characters. The Extended Byte also gave me the tools to solve my problems with endianness. Because it used the Linked List library class I was able to reorder the list much more freely than I could in the string paradigm, moving the bytes to be the right endianess.

The Result

In the end I was able to successfully create a program that generates and writes the appropriate metadata to turn a text file into an image. The final program ended up being a compact 600 lines, a product of months of research, coding and reverse engineering the bitmap format. Along the way I went through numerous refactors, faced dozens of challenges and learned from every one of them.

One of the best lessons I learned was the importance of keeping hand written notes on what the current state of the project, having these notes made it much easier to keep track of where the current issues in the project were on a day to day basis. I plan to continue this practice into my career.

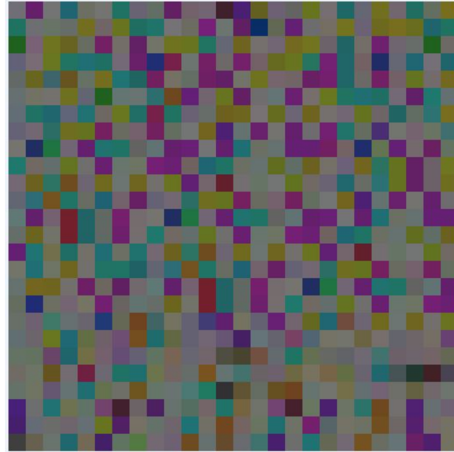
While my program fulfills the key goals that I set out to do, there are a few places that I would look to improve it for the future. I would like to build a graphical user interface (GUI) that would allow less technically adept users to use the application. Ideally I would like to add functionality within that GUI to modify the text and see how the resulting file is changed. Lastly I would like to implement support for other file types, specifically transforming to and from audio files.

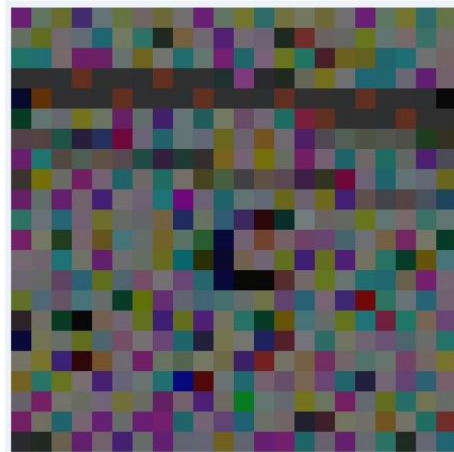
Overall I am very happy with the outcome of the project. The complete code is availible in the appendix but since I plan to continue making changes to this project over time, the most up to date version will always be availible at *www.github.com/tristankj/datavisualizer*

Acknowledgements

I would like to thank my advisor, Dr. Becka Morgan, for all of her support and credit her as the original inspiration for this project. I would also like to thank my friends and family for putting up with me when I rambled on about the zeros and ones of my thesis. I cannot think of a more appropriate way to end, then with the text of my thesis as an image.

And to take it one step further, going even more meta, the code for the program that turns text into images, as an image.





Bibliography

Murray, W. (2007, October 25). Big-Endian, Little-Endian. *Wikimedia Commons*. Wikimedia Foundation, Inc. Retrieved June 1, 2015, from http://commons.wikimedia.org/wiki/File:Big-Endian.svg, little-endian.svg

Verpies (2010 December 2010).BMPfileformat.png. *Wikimedia Commons*. Wikimedia Foundation, Inc. Retrieved June 1, 2015, from http://commons.wikimedia.org/wiki/File:BMPfileFormat.png

Stack Overflow. (n.d.). Retrieved from http://stackoverflow.com/

Stallings, W. (2000). *Computer organization and architecture: Designing for performance* (9th ed.). Upper Saddle River, NJ: Prentice Hall.

Krumin, P. (2013, August 7). ASCII Cheat Sheet. Retrieved from http://www.catonmat.net/blog/ascii-cheat-sheet/

Adams, D. (1980). *The hitchhiker's guide to the galaxy*.

BMP file format. (n.d.). Retrieved from http://en.wikipedia.org/wiki/BMP_file_format

Appendix: The Code

```
/**
 * Main class, the files created appear in the main folder,
 * there are sample text files in the testData folder.
 * @author Tristan
 */
package visualizer;

public class Main {

	public static void main(String[] args) throws Exception {


		String pathOfTextFile = "testData/javaTest.txt";
		//String pathOfTextFile = "testData/thesisRoughDraft.txt";

		String nameOfOutputFile = "Output_" + System.currentTimeMillis()+".bmp";

		Parser par = new Parser();

		TextFile text = new TextFile(pathOfTextFile);
		par.createBmpFileFromData(text);
		par.write(par.modifiedData, nameOfOutputFile);



		//This version of Main uses arguments from command line
		/**
		Parser par = new Parser();

		TextFile text = new TextFile(args[0]);
		par.CreateBmpFileFromData(text);
		par.write(par.modifiedData, args[1]);
		**/
	}

}
--------------------------------------------------------------------------------------------------------
public final class Constants {
```

```
	public static final int BMP_HEADER_SIZE = 14;
	public static final int DIB_HEADER_SIZE = 40;
	public static final int BITS_PER_PIXEL = 24; //3 bytes
	public static final char NULL_CHARACTER = '\0';
	public static final String PRINT_RESOLUTION = "130B";
	public static final Byte[] byteLiteral = { 0b00000000, 0b00000001,
0b00000010, 0b00000011, 0b00000100, 0b00000101, 0b00000110,
			0b00000111, 0b00001000, 0b00001001 };

}
```

```
/**
 * The purpose of this class is to give me more control over the way that
 * I am able to process and manipulate bytes.
 * @author Tristan
 *
 */

package visualizer;

import java.nio.ByteBuffer;
import java.util.Arrays;
import java.util.Iterator;
import java.util.LinkedList;

public class ExtendedByte {

	public LinkedList<Byte> data;

	public ExtendedByte() {
		data = new LinkedList<Byte>();
	}

	/**
	 * Initializes extended byte with value of input and adds trailing zeros to
	 * match the requested Length
	 *
	 * @param input
	 * @param requiredByteLength
	 */
	public ExtendedByte(int input, int requiredByteLength) {
		data = new LinkedList<Byte>();
```

```
		byte[] temp = toByteArray(input);
		for (byte b : temp) {
			if (b != Constants.byteLiteral[0])
				data.add(b);
		}
		while (data.size() < requiredByteLength)
			data.add(Constants.byteLiteral[0]);

		arrayListToLittleEndian();	}

	/**
	 * Empty Bytes Creates Extended Byte of specified size Initialized to be
	 * zero.
	 *
	 * @param emptyBytesNeeded
	 */
	public ExtendedByte(int emptyBytesNeeded) {
		data = new LinkedList<Byte>();
		while (data.size() < emptyBytesNeeded) {
			data.add(Constants.byteLiteral[0]);	}	}

	/**
	 * Makes an extended byte representing the hex input.
	 *
	 * @param hexString
	 * @param totalBytesNeeded
	 */
	public ExtendedByte(String hexString, int totalBytesNeeded) {
		data = new LinkedList<Byte>();
		while (hexString.length() >= 2) {
			data.add(Byte.decode("#" + hexString.charAt(0)
					+ hexString.charAt(1)));
			hexString = hexString.substring(2);	}
		while (data.size() < totalBytesNeeded)
			data.add(Constants.byteLiteral[0]);

		arrayListToLittleEndian();	}

	public byte[] getData() {
		Byte[] temp = new Byte[data.size()];
		int index = 0;
		while (index < temp.length) {
			temp[index] = data.get(index);
```

```
            index++;          }
        return unboxByteArray(temp);  }

    public static byte[] convertBMPMetadata(LinkedList<ExtendedByte> input) {
        int size = fromLittleEndianByteArray(input.get(4).getData());
         // extendedbyteholding file
        byte[] temp = new byte[size];

        ExtendedByte eb;
        int index = 0;
        while (input.peek() != null) {
            eb = input.poll();
            while (eb.data.peek() != null) {
                temp[index] = eb.data.poll();
                index++;          }          }
        return temp;     }

    /**
     * Translates the values of extended bytes storage to be little endian.
     */
    public void arrayListToLittleEndian() {
        Iterator<Byte> it = data.iterator();
        int startingSize = data.size();
        while (it.hasNext()) {
            Byte b = it.next();
            if (b.byteValue() == Constants.byteLiteral[0]) {
                it.remove();     }          }
        while (data.size() < startingSize)
            data.add(Constants.byteLiteral[0]);       }

    public String toString() {
        StringBuilder sb = new StringBuilder();
        int i = 0;
        while (i < data.size()) {
            sb.append(data.get(i) + " ");
            i++;     }
        return "Decimal Value: " + fromLittleEndianByteArray(getData());
    }

    public void printArray() {
        System.out.println("ArrayList Values: " + Arrays.toString(getData())); }

    /**
```

```
 * Because Java apparently isn't smart enough to figure out how to un-box
 * Bytes into bytes.
 *
 * @param Array
 *          to be Unboxed
 * @return A byte[] thats equivalent to the input
 */
private byte[] unboxByteArray(Byte[] input) {
        byte[] temp = new byte[input.length];
        for (int i = 0; i < input.length; i++) {
                temp[i] = input[i].byteValue();
        }
        return temp;
}
public static byte[] toByteArray(int value) {
        return ByteBuffer.allocate(4).putInt(value).array();
}
public static int fromByteArray(byte[] bytes) {
        if (bytes.length < 4) {
                byte[] temp = new byte[4];
                Arrays.fill(temp, Constants.byteLiteral[0]);
                for (int i = 3; i < bytes.length; i--) {
                        temp[i] = bytes[i];
                }
                bytes = temp;    }
        return ByteBuffer.wrap(bytes).getInt(); }

public static int fromLittleEndianByteArray(byte[] bytes) {
        if (bytes.length < 4) {
                byte[] temp = new byte[4];
                Arrays.fill(temp, Constants.byteLiteral[0]);
                for (int i = 3; i < bytes.length; i--) {
                        temp[i] = bytes[i];
                }
                bytes = temp;
        }
        return bytes[3] << 24 | (bytes[2] & 0xFF) << 16
                        | (bytes[1] & 0xFF) << 8 | (bytes[0] & 0xFF);           } }
```

```
/**
 * Class that handles the creation of new metadata          **/
```

```
package visualizer;

import java.io.BufferedOutputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.nio.charset.StandardCharsets;
import java.util.LinkedList;

public class Parser {

	byte[] textBytes;
	byte[] modifiedData;
	LinkedList<ExtendedByte> metaData;
	ExtendedByte eb = new ExtendedByte();

	public Parser() {
		// System.out.println(System.getProperty("user.dir"));
	}
	/**
	 * Transforms a string into a byte array that has been encoded as ASCII
	 * @param str Input to be transformed
	 * @return byte array of str
	 */
	public static byte[] stringToBytesASCII(String str) {
		byte[] b = str.getBytes(StandardCharsets.US_ASCII);
		return b;
	}

	/**
	 * Master method for creating and adding each part
	 * of the bmp file to the list that holds the new file.
	 * @param textToBeConverted
	 */
	public void createBmpFileFromData(TextFile textToBeConverted) {
		createBmpHeader(textToBeConverted);
		createDibHeader();
		addTextDataToNewImage();
	}

	private void createBmpHeader(TextFile textInput) {
		textBytes = textInput.getTextData();
```

```
		int modifiedLength = textBytes.length + Constants.BMP_HEADER_SIZE
					+ Constants.DIB_HEADER_SIZE;
		metaData = new LinkedList<ExtendedByte>();

		// Generate BMP Header metadata
		metaData.add(new ExtendedByte("424D", 2)); // magic number for BMP files
		metaData.add(new ExtendedByte(modifiedLength, 4));

		metaData.add(new ExtendedByte(2)); // unused field
		metaData.add(new ExtendedByte(2)); // unused field
		metaData.add(new ExtendedByte(Constants.BMP_HEADER_SIZE
					+ Constants.DIB_HEADER_SIZE, 4)); // offset of the pixel
array
	}

	private void createDibHeader() {
		// Generate DIBHeaderMetadata
		metaData.add(new ExtendedByte(Constants.DIB_HEADER_SIZE, 4));

		// Calculate the size of the image
		double totalPixels = textBytes.length / Constants.BITS_PER_PIXEL;
		int imageSize = (int) Math.sqrt(totalPixels); // find dimensions of

		// image, rounding down
		metaData.add(new ExtendedByte(imageSize, 4)); // width of the image
		metaData.add(new ExtendedByte(imageSize, 4)); // height of the image

		metaData.add(new ExtendedByte(1, 2)); // color Planes Used
		metaData.add(new ExtendedByte((Constants.BITS_PER_PIXEL), 2));

		metaData.add(new ExtendedByte(4)); // No Pixel Array Compression
		metaData.add(new ExtendedByte(textBytes.length, 4)); // size of Rawbitmap

		metaData.add(new ExtendedByte(Constants.PRINT_RESOLUTION, 4));
		metaData.add(new ExtendedByte(Constants.PRINT_RESOLUTION, 4));

		metaData.add(new ExtendedByte(4)); // number of colors in pallet
		metaData.add(new ExtendedByte(4)); // 0 means all colors are important
	}

	private void addTextDataToNewImage() {
		byte[] tempMetaData = ExtendedByte.convertBMPMetadata(metaData);
```

```
        modifiedData = new byte[tempMetaData.length + textBytes.length];

        int index = 0;
        while (index < tempMetaData.length) {
            modifiedData[index] = tempMetaData[index];
            index++;
        }
        int index2 = 0;
        while (index2 < textBytes.length) {
            modifiedData[index] = textBytes[index2];
            index++;
            index2++;    }    }

    public void write(byte[] input, String outputFileName) {
        // writing binary file.
        try {
            OutputStream output = null;
            try {
                output = new BufferedOutputStream(new FileOutputStream(
                        outputFileName));
                output.write(input);
            } finally {
                output.close();
            }
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();    }    }

    public byte[] getDataRead() {
        byte[] temp = textBytes.clone();
        return temp;    }    }
```

```
/**
 * reads in the text file and generates the text data byte array.
 * @author Tristan
 */

package visualizer;
```

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class TextFile {

	private byte[] textData;

	public TextFile(String filePath) {
		BufferedReader reader = null;
		try {
			File file = new File(filePath);
			reader = new BufferedReader(new FileReader(file));
			StringBuilder sb = new StringBuilder();

			String line;
			while ((line = reader.readLine()) != null) {
				sb.append(line);
			}
			reader.close();
			textData = Parser.stringToBytesASCII(sb.toString());
		} catch (IOException e) {
			e.printStackTrace();	}	}

	public byte[] getTextData() {
		return textData;}

	public void setTextData(byte[] textData) {
		this.textData = textData;		}	}
```