A Ubiquitous Mobile System For Stem Education Enhancement Using Cellular Phone Messaging, Socialnetworking Technology And High Levelcomputation/Visualization

Brandon C. Judd
North Carolina Agricultural and Technical State University

A UBIQUITOUS MOBILE SYSTEM FOR STEM EDUCATION ENHANCEMENT USING CELLULAR PHONE MESSAGING, SOCIAL NETWORKING TECHNOLOGY AND HIGH LEVEL COMPUTATION/VISUALIZATION

by

Brandon C. Judd

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Department: Electrical and Computer Engineering
Major: Electrical Engineering
Major Professor: Dr. Corey Graves

North Carolina A&T State University
Greensboro, North Carolina
2011

School of Graduate Studies
North Carolina Agricultural and Technical State University

This is to certify that the Master's Thesis of

Brandon C. Judd

has met the thesis requirements of
North Carolina Agricultural and Technical State University

Greensboro, North Carolina
2011

Approved by:

Dr. Corey Graves Major Professor	Dr. Alvernon Walker Committee Member
Dr. Christopher Doss Committee Member	Dr. John Kelly Department Chairperson

Dr. Sanjiv Sarin
Dean of Graduate Studies

BIOGRAPHICAL SKETCH

Brandon Judd obtained a Bachelor of Science degree in Electrical Engineering from North Carolina Agricultural and Technical State University in 2007. Throughout his course work he held a particular interest in micro processing applications, embedded system design, IEEE robotics competitions, and the seamless integration of computers in everyday life. During his undergraduate career he also became a member of many notable organizations including: The National Society of Black Engineers, IEEE, Eta Kappa Nu Honors Society, and Alpha Lambda Delta Honors Society. After graduating he worked in the defense industry for two years, where he realized that his true passion wasn't writing technical documents but rather practical applications of Pervasive Computing. After which he abruptly decided to come back to school for a Masters of Science in Electrical Engineering, where he studies under Dr. Corey A. Graves as a thesis candidate.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Judd, Brandon C. A UBIQUITOUS MOBILE SYSTEM FOR STEM EDUCATION ENHANCEMENT USING CELLULAR PHONE MESSAGING, SOCIAL NETWORKING TECHNOLOGY AND HIGH LEVEL COMPUTATION/VISUALIZATION. **(Major Advisor: Corey Graves),** North Carolina Agricultural and Technical State University

There is a dire need for more STEM degrees in America. Recent studies have shown that there is a surge in motivating the youth in STEM areas of research. One key motivator is the use of pervasive computing concepts such as SMS/MMS messaging and social networking. Teens of today send text messages more than ever; in fact research shows that teens (ages 13 to 19) use cell phones to text far more than to actually make phone calls. Thus SMS/MMS messaging presents itself as an excellent segue into STEM motivation and education. Computational/visualization engines, with their broad application development capabilities, can be used as an excellent teaching tool at all levels simply by modifying applications to fit specific course material. Thus the UMSEE system was created.

UMSEE (Ubiquitous Mobile Systems for Educational Enhancement) is a system conceived by Dr. Corey A. Graves that interfaces computational/visualization engines, and SMS/MMS messages. The system allows users to send processing request to a computational/visual engine toolbox (e.g. MATLAB, Octave) via SMS or MMS and receive the processed data back via SMS or MMS (e.g. in the form of JPEG graphs). The use of an SMS/MMS-based social networking service, versus a dedicated email account, was chosen as the interface between SMS/MMS and the computational/visualization engine due to its ease of application development, social relevance, popularity, user-friendliness, and security. In addition to the aforementioned reasons, SMS/MMS-based social networking sites have the potential to ultimately draw the attention of younger people toward STEM learning because of their familiarity with the social networking site.

The development of the UMSEE system using the social networking site managed better versus using a dedicated email account. This was primarily due to the open source development environment of the social networking site. Overall, the social networking site based system was far more robust in terms of security, speed, development, and SMS/MMS messaging.   Furthermore future UMSEE users can easily tailor the computational/visual engine toolbox for specific courses or demonstrations. Based on the increase in pervasive computing in education, the UMSEE has potential to be a major part of education.

CHAPTER 1

INTRODUCTION

1.1 Motivation

There has been an increasing amount of STEM (Science, Technology, Engineering, and Mathematics) graduates in the United States over the past 15 years. Moreover, strategic measures in increasing STEM interest must be taken to ensure that this trend continues. Funding agencies have shown interest in funding STEM initiatives to ensure that the demand for scientist, technologist, engineers, and mathematicians are met in the future (Gehrig, Abrams, Bosley, Conrad, & Kuyath, 2009). Most people enjoy the fruits of the labors due to STEM, such as Playstation 3's, Navigation Systems, and even backup cameras, but much rather not work in the actual fields. To the average person, applying the basic principles of STEM to something they do not deal with in everyday life is very enigmatic, for example understanding that changes in air pressure is used to fly airplanes. However, if you consider an application that is dealt with on a daily basis such as air pressure in a tire, the average person will understand. Clearly people understand things better, when they deal with them everyday. Perhaps STEM interests can be increased by creatively incorporating them in a daily routine.

1.2 Summary of Background

Virtually all STEM education and research can be enhanced by some sort of scientific computational/visual engine (e.g. MATLAB and Octave). The scientific computational/visual engine contains its own high level programming language and is used to create applications containing user interfaces, information handling, and information display (Moler, n.d.). With this wide range of complexity, computational/visual engines can be utilized at pretty much all levels, from graphing equations to graphing the topography of a mountainous region, refer to Figure 1. In the past, two drawbacks of these engines have been their high price tags and licensing issues.

Cellular phone messaging allows an easy capture and exchange of multimedia information. Moreover, the use of cellular messaging allows non-verbal chats between friends just about anywhere, at anytime, and it is a very accessible technology. It is noteworthy, that cellular phone messaging (not calling) is the main mode of communication by the youth of the United States (Lenhart, Ling, Campbell, & Purcell, 2010). Also of importance is the fact that social networking technologies such as Myspace, Facebook, and Twitter, are regularly used by youth and offer a socially relevant way to engage students. In addition, there are a plethora of internet based social networking technologies available, with Facebook being number one, Myspace coming in second, and Twitter in third (Nations, 2010). Computational/Visualization engines are optimized for data processing and are for the most part inaccessible, while cellular phone messaging is optimized for data sharing and are very accessible. Creatively merging these two technologies together yields a novel approach to increasing STEM education and research in youth.

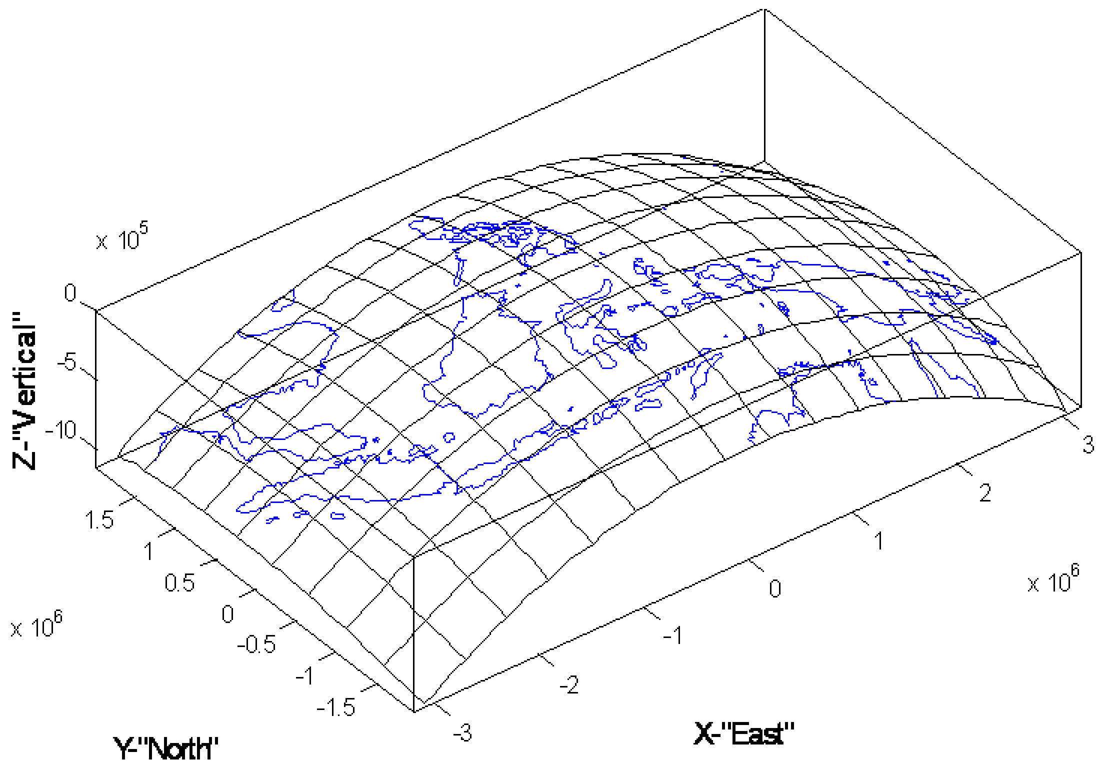


Figure 1: Computation/Visualization Topographic Image (www.google.com)

1.3 Proposed Solutions

The problem with computational/visualization engines is that students and STEM prospects are usually not introduced to these engines until the college level, and due to the steep price tag, not even college students can afford a copy of the software. They usually resort to having to use a copy installed on a school computer in a computer lab. Technology is becoming more ubiquitous and people have ever increasing access to and interest in, cellular messaging, and online social networking. Taking advantage of these technologies allows for an easy segue to engage students in STEM learning. Leveraging the popularity of these technologies is what has motivated the idea of UMSEE (Ubiquitous Mobile Systems for Educational Enhancement). UMSEE is a system conceived by Dr. Corey A. Graves that gives students a pervasive approach to STEM subjects by interfacing a computational/visualization engine to SMS/MMS messages, potentially sparking more interest in STEM education and research. Originally UMSEE was conceived to interface SMS and a computational/visualization engine via a dedicated email account. However, there are problems with using a dedicated email account, most notably the development of parsing schemes. Each service provider (e.g. Sprint, Verizon, AT&T) format their SMS/MMS messages differently. For the most part this is not a problem, the only problem is when trying to develop a standardized application capable of extracting necessary data from the messages by parsing through the data (such as the application discussed here). In order to handle that problem, different parsing schemes need to be developed for each carrier.

Web based SMS/MMS social networking services present themselves as excellent interfaces between the students and the computation/visualization engine. There are various advantages to using such services versus a dedicated email account. For one, such services have open source predefined APIs (Application Program Interface) that allow developers to easily create applications that interface to them ("Twitter API Wiki," n.d.). Since Twitter is an SMS based social network, it already parses the different SMS formats from each service provider, thus there is no need to develop parsing schemes for each carrier. In addition, as suggested by (Gehrig et al., 2009), Twitter is the perfect example of a socially relevant way of engaging students. Twitter is the third most visited social network site (Nations, 2010). The greatest advantage of Twitter is its security. Normal email accounts have to be “popped” each time the user wants to retrieve something. This can pose an issue because some email accounts will lock users out if they pop the server too often within a short amount of time. Twitter uses the Oauth protocol which allows account holders to have very specific access to applications without having to sacrifice the anonymity of their username and password (“Oauth,” 2010). Moreover, Twitter is simply more robust due to the volume of request it can handle and the unlikely probability of the network crashing. With the advent of SMS/MMS web services like Twitter and its open source API, an application can be created to standardize the interface between, mobile devices, and a scientific computation/visualization engine. Within the UMSEE system, this will offer several advantages over an ad-hoc email account-based interface.

CHAPTER 2

BACKGROUND

2.1 Increases in Pervasive Computing and STEM Education

Pervasive Computing, also known as ubiquitous computing, is an area of computer engineering that endeavors to make computing power available anywhere, anytime to meet the growing needs of today's society (Reddy, 2006). A key concept within pervasive computing is middlewear. Middlewear is software that facilitates distributed computing by allowing applications or processes to work together over a given network. Smart devices are pervasive computing devices that are able to acquire and process data about the user's surroundings and environment, improving the users experience (Reddy, 2006).

With the onset of mobile smart device technology and increasingly smaller hardware, pervasive computing is on the rise. Cell phones are becoming the gateway into the pervasive world of tomorrow; they are the easiest way a person can take computing capabilities anywhere he/she goes. Pervasive computing technology is pretty much in every aspect of life. Sky scrapers display weather information, cars automatically parallel park themselves, and in some cases they even automatically stop (Shibu & Jain, 2010). For instance, certain automobiles are equipped with an automatic slow down and stop feature that automatically stops or slows the car down, based on the relative proximately of other objects, this is to help prevent accidents (Shibu & Jain, 2010).

According to the National Science Board, higher education degrees in STEM areas are rising (*Science And Engineering Indicators 2010*, 2010). A recent study shows that there has been a steady increase in science and engineering degrees over the past 15 years. More specifically, the National Science Board's study discussed a new high of science and engineering degrees in 2007, close to half a million. Based on the same study, network infrastructure was one of the key drivers for on campus research. This infrastructure included the pervasive use of laboratory instrumentation, which fueled research in STEM. Likewise, grade school curriculums are reforming to include STEM development in education. Texas state high schools have been awarded more the 180 million dollars from a collection of government and private sector contributors in an effort to reform the teaching environment and aiming it more toward STEM concepts (Fontenot, Chandler, Talkmitt, & Sullivan, 2008). This radical change in curriculum is an example of how grade schools are moving towards STEM education. These two increasing trends suggest that pervasive computing and STEM education as the key to technology's future.

2.2 Motivating the Young in STEM

Research shows that the primary approach to motivate STEM in youth is to introduce STEM principles with the use of something already familiar to them. For instance, the IEEE Transactions on Education journal contained an article describing ways to introduce students to semiconductor technology by shooting ball-bearings into jelly to imitate ion implantation and potato stamping to demonstrate the layered structures in silicon chips (Magill & Roy, 2010). Another example of using familiar concepts is how Wichita State University uses LEGOs, a longtime favorite toy amongst children, to create a STEM learning environment (Whitman & Witherspoon, 2004). The university holds LEGO Mindstorm robotic competitions in which students of all grades use STEM and programming techniques to build robots to specification. The university believes that introducing STEM principles by the use of something students are used to, garners a greater appreciation for STEM, and behooves society overall (Whitman & Witherspoon, 2004).

2.3 Scientific Computational/Visualization Engines

Scientific Computation/Visualization Engines have the potential to enhance STEM education and research. These engines offer high level programming languages which makes learning how to use them extremely easy. For instance creating two variables and initializing them with values is as easy as Variable1 = 10, and Variable2 = 2. To add to this ease of use, they contain an aesthetic interface, which makes them easy to operate. Due to their high level language and aesthetic interface, computational/visualization engines can offer a wide range complexity. Programs can be created that do something as simple as add two numbers together or something as complex as showing the intensity heat map of a given image, refer to Figure 2. As a result of its wide range of complexity, these engines can be introduced at all school levels. In the e-book, “Experiments with MATLAB”, author Cleve Moler offers numerous pre-written MATLAB applications aimed at high school students (Moler, n.d.).

The applications vary from Tic Tac Toe to using Fibonacci numbers in solving linear equations using MATLAB.

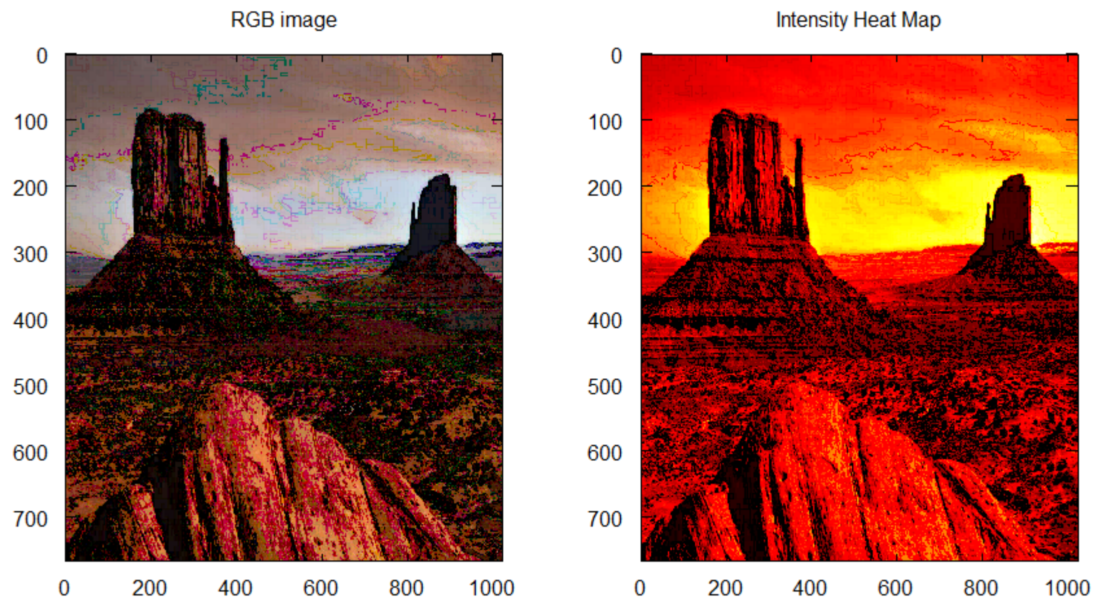


Figure 2: Intensity Heat Map of a Given Image

One major drawback of computational/visualization engines are price. While the creators of MATLAB, a computational/visualization engine used by most universities, offer a watered down student version, the true price of MATLAB goes far beyond the licensing budgets of most public middle and high schools. This high price tag makes the engines inaccessible to most students seeking to use it. Furthermore, in order to use the computational/visualization engines, a desktop or laptop computer is needed to run the actual application. This further limits its accessibility to prospective students. One way to circumvent the high price tag of these engines is by using a free, open source computational/visualization engine such as Octave. Octave is a computational/visualization engine that is functionally equivalent to MATLAB ("GNU Octave Documentation," n.d.). Octave is freeware with an open source environment, which allows users to programmatically support the software, by adding/amending files. With the use of the freeware Octave, users do not face licensing issues as they would with using MATLAB.

2.4 Cellular Phone Messaging

Cellular phone messaging such as SMS or MMS is a key motivator of STEM. SMS (Short Messaging Service), also known as text messaging, is a messaging protocol used to send messages between a mobile phones (Firdaus bin Haji Sidek, 2010), while MMS (Multimedia Messaging Service), which advanced from SMS, is a protocol that transmits multimedia (such as pictures, audio, and video) (Mostafa., 2002). The original idea of SMS was conceived during the mid 1980's and did not gain popularity until late 1990's (Firdaus bin Haji Sidek, 2010). MMS was designed to allow mobile phone consumers the ability to send multimedia with their text (Mostafa., 2002). SMS/MMS messaging is an easy means of exchanging multimedia in a pervasive manner. Most often short code services, which are five digit numbers that users send text messages to, are used to vote, donate money, use social networking, and even download data (Fehrenbacher, 2007).The primary drive behind the use of short codes is the ease of use. It is easier for someone to donate, vote, or download something when they can do it with their mobile phone, which means it can be done virtually anywhere and anytime. Similarly, this ease of use of texting short codes can be applied to SMS/MMS use in

STEM. Studies have shown that text messaging is something the youth in the U.S. are highly familiar with. A study done by Pew Research center shows that teens spent most of their day sending text messages, and that the average teen sends about 50 texts per day (Lenhart et al., 2010), Figure 3 shows the mean and medium number of texts sent per day by teen texters, according to age and sex.

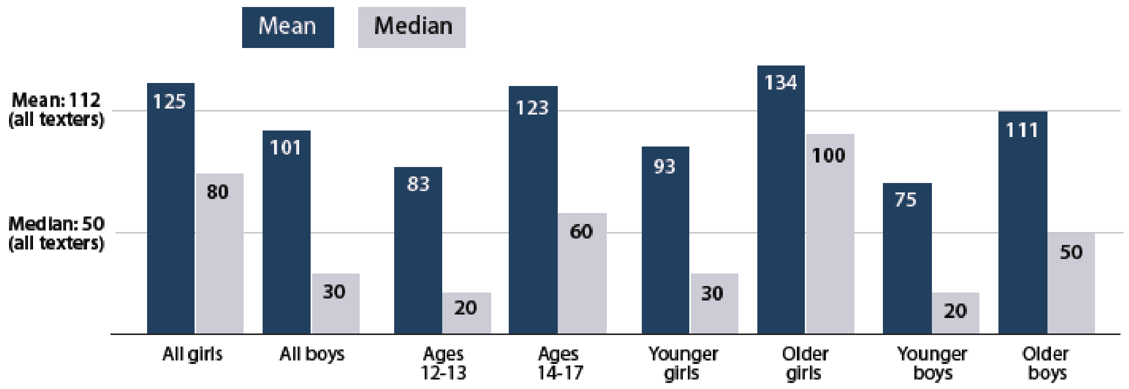


Figure 3: Typical Number of Texts per Day (Lenhart et al., 2010)

Texting is a tool the youth is highly familiar with, therefore it fares as a great segue into STEM fields. Not only are the youth very familiar with texting, but may also offer them easy access to STEM information. Based on a study done after the 2008 presidential election, text messaging amongst younger adults (18 – 29) was far more popular (85% - 65%) than in adults over the age of 30 (65% and below) (Kiyohara, 2009). Table 1 shows mobile communication activity by age. The upper section of Table 1 shows those who have a cell phone or personal data assistant who have ever done one of listed activities, while the lower section shows those who have a cell phone or personal data assistant who do one of the listed activities on a typical day. Moreover, the Obama campaign purposely used text messaging to target the younger generation.

Table 1: Mobile Communication Activity by Age (Kiyohara, 2009)

Activity	18 – 29	30 – 49	50 – 64	65+
Send or receive text messages	85%	65%	38%	11%
	60%	32%	14%	2%

2.5 Social Networking Technologies

Social networking technologies are websites that offer an online based social networking experience. Social networking websites hit the scene in the early 2000’s and are now a phenomenon. The social networking site with the most activity is Facebook, with MySpace in second, and Twitter in third (Nations, 2010). Social networking technologies are the perfect example of a socially relevant way of engaging students. Social networking technologies are regularly used by youth. Table 2 below illustrates the overall demographics of social network users. The biggest attraction to social networking technologies is the fact that they are free to use, and they are a plethora of internet based social networking sites available.

Table 2: Demographics of Social Network Users

Age	Percentage
18 – 24	75
18 – 34	57
35 – 44	30
45 – 54	19
55 – 64	10
Over 65	7

2.6 Merging Computational/Visualization Engines with Cell Phone Messaging

Computational/visualization engines have the ability to enhance STEM education and research, yet are largely inaccessible to most students. Additionally, cellular phone messaging is an easy way to exchange multimedia information, and is a cheap and easily accessible technology. By taking advantage of cellular phone messaging, and computational/visualization engines, a system aimed at giving STEM prospects the ability to learn STEM concepts can be created UMSEE (Ubiquitous Mobile Systems for Educational Enhancement) is a system proposing to use SMS/MMS (cellular phone messaging) to remotely interact with a given computational/visualization engine. UMSEE merges cellular phone messaging and computational/visualization engines by using SMS/MMS messaging to remotely interact with a computational/visualization engine. This type of approach facilitates an anywhere/anytime advance in computer aided STEM learning.

CHAPTER 3

DEVELOPMENT OF UMSEE

3.1 UMSEE System Description

UMSEE is a system that is used to promote the pervasive use of a computational/visualization engine such as MATLAB or Octave (freeware that is functionally equivalent to MATLAB). Currently the UMSEE system takes advantage of Octave. UMSEE is designed to be used in a class setting in which the instructor creates a course specific Octave module he/she wants the students to be able to interact with during class using the created toolbox. The Octave environment will be displayed to the entire class via a projector during the execution of the module. The Octave module will consist of a few Octave functions that require at least one type of multimedia input (text or image) data, and return at least one type of multimedia output.

The interface between the students SMS/MMS request and the instructor’s educational module is the third party application. MyChatLab is the name of the interface application. It simply extracts all request sent by the students/users and stores them in a shared database of the education module. A student will interact with a module by using SMS/MMS messaging from any cell phone that is enabled to do so. The student texts requests to a dedicated email account. The instructor has the option of generating pre-defined functions that the students can use when texting in multimedia.

3.2 Using Dedicated Email Account

The student communicates to the instructor through the dedicated account set up for the UMSEE system. The dedicated email account is controlled by the abovementioned third party application, MyChatLab. Before the student can send requests to the UMSEE system, they need to register themselves with the system by texting the key word reg, followed by their ten digit number and the carrier. Once the student is registered the communication can be begin. Figure 4 helps illustrate a given teaching scenario, where a student creates their own programming file (also known as M-file), for the UMSEE system.

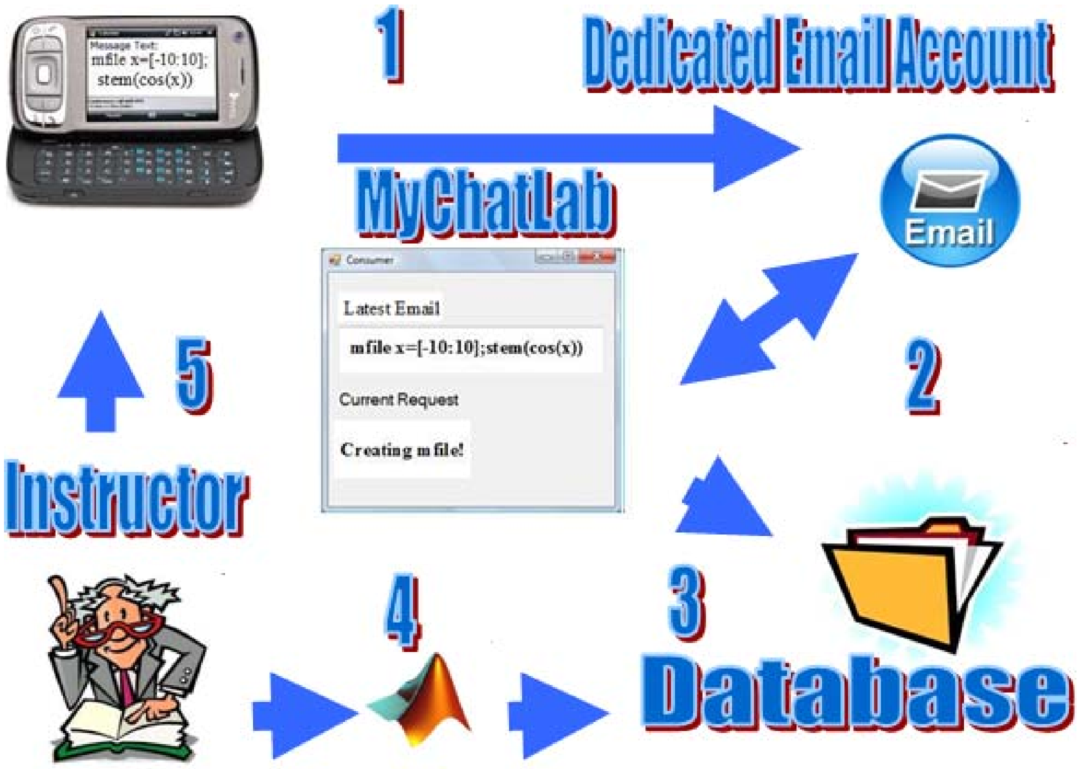
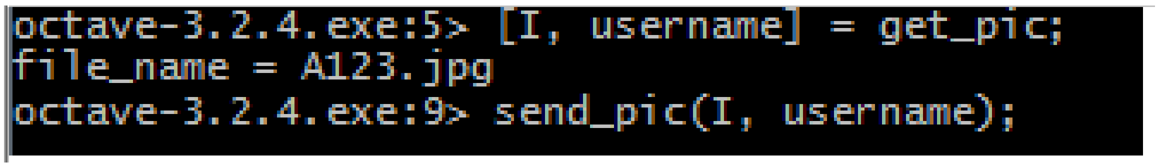


Figure 4: UMSEE Teaching Scenario

First, the student texts in a request, to the dedicated email account UMSEE@hotmail.com, with the key word “mfile” followed by the programming text for the file they want to create. Second the MyChatlab application extracts the requests send by the student from the dedicated email and third it stores the request in the UMSEE database. The fourth step is where the instructor uses the MyChatLab toolbox and pulls the earliest request sent using the Octave functions runmfile or get_pic. The runmfile function runs the earliest sent M-file and sends the results of the M-file back to the student who originally sent it. The get_pic function gets the earliest image sent along with its corresponding M-file. The function displays the image, and processes it based on what the corresponding M-file is programmed to do. Finally the get_pic function returns the processed image, and the username of the student who sent the image as variables. The fifth and final step in the process is the instructor sending the results of the personalized program back to the student’s cell phone via SMS/MMS; this is done using the send_pic function. Figure 5 shows the Octave user interface, and how the get_pic and send_pic functions are used. The get_pic function returns the image sent by the user as a variable along with the username of the student who sent the image. The send_pic function sends the processed image back to the student with the image and username as inputs.



```
octave-3.2.4.exe:5> [I, username] = get_pic;
file_name = A123.jpg
octave-3.2.4.exe:9> send_pic(I, username);
```

Figure 5: Octave User Interface

There are problems with using a dedicated email. Extracting the SMS/MMS message is somewhat of a cumbersome process. Service providers (e.g. Sprint, Verizon, AT&T) format their SMS/MMS messages differently. Thus there is no standard data extraction method for messages of different carriers. Different parsing schemes need to be developed for each carrier. Also, email accounts limit the frequency that third party applications, such as MyChatlab, can pull information from their servers. The greatest issue with using a dedicated email account is the security.

The MyChatlab application interacts with the dedicated email using a Telnet connection, a web based protocol used to connect machines over a given network (Mahmood, 2003). Telnet has known security issues that have been around for quite a while. One the most important security vulnerabilities of telnet is the possibility of having the password to the account sniffed out (Mahmood, 2003). Also telnet connections can be hijacked and used as a server instead of a service by an external hacker seeking vulnerabilities (Mahmood, 2003).

3.3 Incorporation of Twitter

An alternative to using a dedicated email account is to use a dedicated Twitter account instead. Twitter is an SMS based social networking technology that can be used to retrieve data send by the student via SMS (“Twitter API Wiki,” n.d.). Therefore no parsing schemes need to be developed because it will be handled by Twitter. Moreover, Twitter is the perfect example of a socially relevant way of engaging students. In addition to the SMS abilities, Twitter has its very own API (Application Program

Interface), and an online developer's community intended to foster an open source environment, that allows new comers to develop third party Twitter applications, such as MyChatLab, while simultaneously improving the Twitter API ("Twitter API Wiki," n.d.). Twitter also uses a security protocol known as Oauth. Oauth is a protocol that allows users of Twitter to develop and use third party applications without having to reveal username and password information ("Oauth," 2010). Figure 6 illustrates the basics of how Oauth works. Ouath works by using request tokens and access tokens, instead of using usernames and passwords. The three entities involved in the Oauth process are the user, consumer application, and the service provider. In the case of the UMSEE system, the user is the student, the consumer is the MyChatLab application itself, and the service provider is Twitter. Basically the user needs to access certain things from the dedicated Twitter account that will require a username and password, and Oauth is used to circumvent the need to have a username and password.

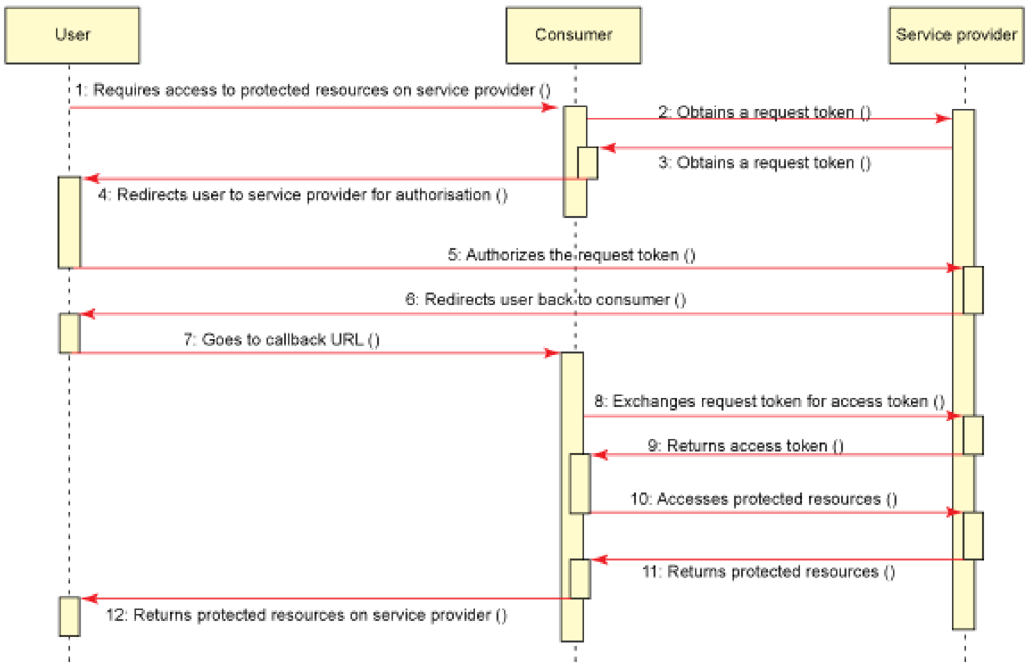

User
Consumer
Service provider
1: Requires access to protected resources on service provider ()
2: Obtains a request token ()
3: Obtains a request token ()
4: Redirects user to service provider for authorisation ()
5: Authorizes the request token ()
6: Redirects user back to consumer ()
7: Goes to callback URL ()
8: Exchanges request token for access token ()
9: Returns access token ()
10: Accesses protected resources ()
11: Returns protected resources ()
12: Returns protected resources on service provider ()

Figure 6: Ouath Security Process ("Oauth," 2010)

3.4 Using Twitter with UMSEE

The student communicates to the instructor through the dedicated Twitter account set up for the UMSEE system. The dedicated Twitter account is controlled by MyChatLab. In order for the student to send in any kind of input, the student themselves must have a Twitter account. Moreover, the student and the dedicated Twitter account must be following each other. However the student must follow the dedicated Twitter account first, then the Twitter account will automatically follow the student. To do this, the UMSEE system takes advantage of an available third party application called Socialoomph that automatically accepts followers and follows them back (www.socialoomph.com). Next the student simply links their Twitter account to their cell phone. Once the student and the UMSEE Twitter account are following each other, communication can begin. As with the dedicated email account, the student must first register with the system by sending a text message containing the key word “reg” followed by their ten digit number and the carrier. The way a student sends in text input is through direct messaging. Direct messaging is Twitter’s way of allowing users to send private messages between each other. To do so from a cell phone simply text to the short code 40404 (Twitter) and include d PSEENCAT and the message.

There are two things students can do with the UMSEE system; create M-files (a file that contains the students personalized program), and send images with M-files. To create an M-file the student sends a direct message to the dedicated Twitter account with the key work “mfile” followed by the text for the M-file. In order to receive images from the students, the UMSEE system takes advantage of another very popular third party application; Twitpic. Twitpic is extremely easy to set up, simply go the Twitpic website (www.twitpit.com) while logged into the Twitter account and it asks the user if he/she wants to log in with their Twitter account. Once this happens, Twitpic assigns the user an individual email address that allows him/her to send in images. The student simply needs to take a picture with their phone and send the image to the corresponding individual Twitpic account. The image will then show up as a link in the user’s Twitter update. In order to send an image to the UMSEE account the student must mention the UMSEE account (PSEENCAT) in their subject line of the image to be sent. In Twitter a mention is when one Twitter user includes another user in his/her updates. This is done by including the @ symbol followed by the user name of the account that person wants to mention. Not only can the student send in an image, he/she can also include a corresponding M-file to that image. To send an image the student sends an MMS message from their cell phone to their Twitpic account, with @PSEENCAT in the subject line. The student also has the option of creating an M-file associated with that image. To create an associated M-file with an image, in the subject line, following @PSEENCAT, the user places two question marks to denote the start of the M-file text, inserts the text, and two question marks after the text to denote the end of the M-file text. For instance the instructor may have a predefined function that shows the intensity heat map of a given image (refer to Figure 2), in order to use that function the user would include "??heatmap(I)??" in the text of the image message. The UMSEE system automatically creates a variable I that allows the user to programmatically process the image by referring to the image as I, refer to Figure 7.

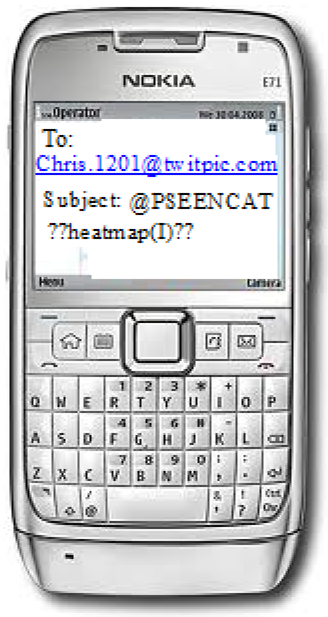
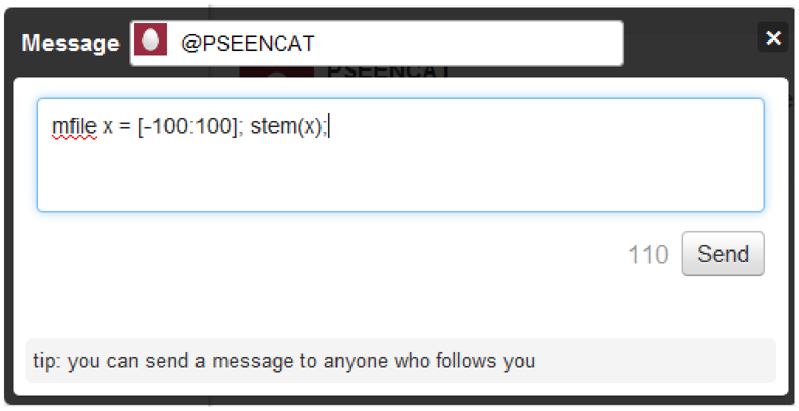

To:
Chris.1201@twitpic.com
Subject: @PSEENCAT
??heatmap(I)??

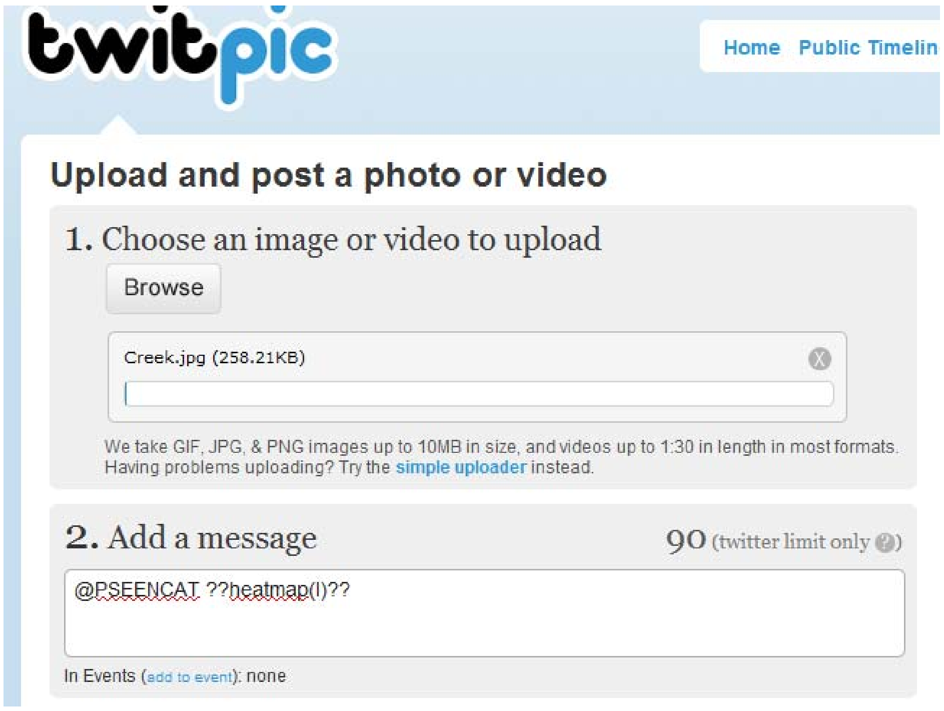
Figure 7: Sending M-file with Image

Along with creating M-files and sending images via SMS/MMS messages, the user also has the option of doing those same things via the Twitter and Twitpic website interfaces. The process is still the same; however instead of using a cell phone, the user sends everything through the website. For instance, to create an M-file, the user sends a direct message to PSEENCAT from the message link on the Twitter website with the key word "mfile" followed by the text to place in the M-file. Likewise, an image with an associated M-file can be sent through the Twitpic website interface where the user mentions PSEENCAT in the text field by including "@PSEENCAT" accompanied by two question marks to denote the start of the M-file text, and two question marks to denote the end of the M-file text, refer to Figure 8 to create a M-file and refer to Figure 9 to send an image with an M-file using the website interface.



```
mfile x = [-100:100]; stem(x);
```

Figure 8: Creating M-file from Twitter Website (www.twitter.com)


Upload and post a photo or video

1. Choose an image or video to upload

Creek.jpg (258.21KB)

We take GIF, JPG, & PNG images up to 10MB in size, and videos up to 1:30 in length in most formats.
Having problems uploading? Try the simple uploader instead.

2. Add a message 90 (twitter limit only ?)

@PSEENCAT ??heatmap(I)??

In Events (add to event): none

Figure 9: Send Image with M-file from Twitpic Website (www.twitpic.com)

3.5 Competition

The UMSEE system is not the first system to access a computational/visualization engine pervasively, there is some competition. One of the more popular ways of doing so is the MATLAB application for the iPhone: MATLAB Mobile. MATLAB Mobile is a desktop application for the iPhone that connects users to MATLAB remotely (“MATLAB Mobile,” n.d.). It is a relatively crafty application that allows command line access, access to a given workspace, and even the ability to view MATLAB Figures. However, there are shortcomings of MATLAB Mobile. MATLAB Mobile lacks in its ability to support the MATLAB editor. This means that users of MATLAB Mobile cannot create M-files. Another weakness of MATLAB Mobile is that the user has to have a network connection to communicate with MATLAB. Furthermore, the user must have a copy of MATLAB installed on a desktop or laptop. UMSEE uses SMS/MMS messaging to lessen the shortcomings of MATLAB Mobile. Since the MyChatLab application interfaces SMS/MMS with Octave, it has direct access to its database and the editor, which means it can create M-files (and run M-files if need be). The UMSEE system also capitalizes on the GNU open source freeware, Octave, which is free to the public, which means a user of UMSEE does not have to worry about licensing issues. Lastly, the UMSEE system takes advantage of SMS/MMS messaging to circumvent the need to have a network connection just to use the system.

CHAPTER 4

EXPERIMENTAL SETUP AND RESULTS

4.1 Explanation of Setup

In order to test the UMSEE system an experiment was devised to highlight the performance difference between a dedicated email account based system and the dedicated Twitter based system. The experiment consisted of recording the time it took for a message (text, or image) to go from a particular cell phone to the MyChatlab application, and vice versa. The experiment included a total of 60 trials, 30 trails for the Twitter based system and 30 trails for the dedicated email based system. There were 15 trails for sending M-files to MyChatlab, 15 trials for sending images to MyChatlab. The last 15 trials test the time it took for information to go from the MyChatLab toolbox to the user's cell phone. Each trail used the same exact M-file for the M-file trials, and the same image for the image trials. The reason for using the same exact files is to reduce variability in the results.

Table 3 and 4 below illustrate the results of the dedicated email based system and the dedicated Twitter account based system respectively. In Tables 3 and 4 the "phone to app time" denotes the average time after which the cell phone actually sent the message to the time the M-file was created within the MyChatLab database. The "app to phone time" signifies the average time at which the MyChatLab toolbox sent the data to the time the cell phone received that data, not the time it took for the data to download to the phone, this is also an average. In addition, the "app to phone time" is also measured.

This is the same for both the Twitter based account and the dedicated email account because sending data back to a user's phone is independent of which system the UMSEE based on. To record these times a stopwatch was started at the starting point of each respective criteria and stopped at the end. During the experiments, the MyChatLab application was altered to notify the user when an M-file is created or an image is received so that the end of the criterion time is known. The length of the message overhead is the number of characters needed to create a message, such as creating M-files, or sending images. The length of the address is the number of characters of the address to where the message is going. The success rate is simply the ratio of successful message completions to failures. A failure can range from an account lock out, due to rate limiting, or a website server issue (e.g., the Twitter website being overcapacity). A success is simply a received image and/or M-file. Lastly, the initial setup steps are the steps that need to be completed prior to being able to use the UMSEE system.

Table 3: M-file Experiment Results

Criteria	Twitter Based System	Email System
Phone To App Time	12.9 seconds	54.4 seconds
App To Phone Time	23.25 seconds	
Length of Message Overhead	14 characters	5 characters
Length of Address	5 characters	17 characters
Success Rate	100%	53.3%
Initial Setup Steps	4	1

Table 4: Image Experiment Results

Criteria	Twitter Based Account	Dedicated Email Account
Phone to App Time	25 seconds	50.2 seconds
App to Phone Time	23.8 seconds	
Length of Message Preamble	14 characters	4 characters
Length of Address	17 characters + length of username	25 characters
Success Rate	100%	46.6%
Initial Setup Steps	5	1

4.2 Results

Tables 3 and 4 show the testing results. The M-file results in Table 3 show a large difference in the "Phone to App Time" between the Twitter based system and dedicated email based system. There was an average of 23.25 seconds in the "App to Phone Time". The message overhead for the Twitter based system has 14 characters, "@PSEENCAT mfile", while the overhead for the dedicated email based system is only five characters, "mfile". The address field of the M-file has a length of five characters for the Twitter based system (40404), whereas the email account based system has 17 characters (UMSEE@hotmail.com). The success rate for the Twitter based system was 100%, and 53.3% for the email account based system, which means only eight of out the 15 trails were a success with the email based system. This was largely due to lock outs from the hotmail server because of rate limiting. Lastly, there are four initial setup steps needed to use the Twitter system. Those steps are: (1) create a Twitter account, (2) connect that account to an SMS/MMS enabled cell phone,(3) follow the PSEENCAT Twitter account, and (4) register with the UMSEE system. The email based system only has one initial setup step, which is to merely register with the system.

Table 4 displays the image experiment results. The "Phone to App Time" for the Twitter based system is almost half the time for the email based system. The average "App to Phone Time" is 23.8 seconds, which is extremely close to the "App to Phone Time" for the M-file results. The lengths of the message overhead are the same as in the M-file results except for the dedicated email, which is one character less. To create an associated M-file with an image using the dedicated email system, the user just includes four question marks along with their text, two at the beginning of the text and two at the end of the text. The Twitter based system's address length changes when sending images. In order to send an image to the Twitter based system, the user sends the image to their corresponding Twitpic account email address. The minimum length of a Twitpic email address is 17 characters, five characters for a personalized pin plus a dot, and 12 characters for @twitpic.com. A typical Twitpic email address is USERNAME.1234@twitpic.com. The success rate for the dedicated email based system decreased to 46.6%, denoting one less successful image extraction. Again this is mostly due to rate limiting. Finally, there is one more step needed in the initial setup steps for the Twitter based system, which is to create a Twitpic account prior to registering with the UMSEE system.

Table 5 shows the qualitative results of the experiment. The "Ease of Development" criterion represents how easy it was to develop the MyChatlab application with that corresponding system base. "Accessibility" refers to how accessible the server was during testing of the application. “Service Provider Handling” means how well it could handle different service providers for the same file (i.e. M-file or image file). “Security” indicates how secure the connection was. In order to qualify these criteria, “Excellent”, “Good”, “Fair”, and “Poor” were used. Where “Excellent” denotes virtually no issues, “Good” stands for very few issues, “Fair” symbolizes more issues than desired yet still plausible, and “Poor” signifies that application is unable to handle that given criterion. As seen in Table 5, the Twitter based system received “Good” and above in every criterion except “Accessibility”, where it received a “Fair”. The primary reason for this mark is due to the fact that the Twitter server would periodically be down due to overcapacity. The email based account only received “Good” in its accessibility. During testing the email account server never went down but was frequently inaccessible due to account lock outs because of accessing the server too often in a short amount of time. Additionally, the MyChatLab application based on a dedicated email account was not easy to develop, nor could it handle many service providers. The most important criterion in Table 5 is the “Security”. As mentioned earlier the Twitter based system uses the Oauth protocol, which is highly developed and secure. The email account based system uses a telnet connection which can easily be “sniffed out” or even completely taken over by internet hackers (Mahmood, 2003).

Table 5: Qualitative Experimental Results

Criteria	Twitter	Email Account
Accessibility	Fair	Excellent
Service Provider Handling	Excellent	Poor
Security	Excellent	Fair
Accessibility	Fair	Excellent

Testing of the two versions of the UMSEE system confirms that the Twitter based system is the system of choice in terms of how long it takes to receive data. Table 3 shows that the "Phone to App Time" in creating M-files for the email based system was on average 4 times slower, also Table 4 demonstrates that the "Phone to App Time", when sending images, was twice as fast with the Twitter based system. One advantage of the email based system is the amount of characters needed to create an M-file or image file, the shorter the message overhead, the easier it is to interact with the UMSEE system. In addition, Table 5 shows that the email based system only needs one initial setup step to create an M-file or send image, while the Twitter based system needs four and five steps respectively. Having more initial setup steps completes using the Twitter version of UMSEE. The defining criterion for either system was its success rate. Tables 4 and 5 verify each systems success rate, with the Twitter based system 100% reliable in creating M-files and sending images. The Twitter based system never failed in creating a users M-file or receiving their image.

CHAPTER 5

CONCLUSION

5.1 Summary

Funding agencies have shown interest in funding STEM initiatives to ensure that the demand for scientisst, technologists, engineers, and mathematicians are met in the future. Moreover, interest in STEM may be increased by creatively and pervasively incorporating its principles into a daily routine. Scientific computational/visualization engines (e.g. MATLAB or Octave) can enhance STEM education and research with their easy to learn programming languages, aesthetic interface, and wide range of complexity. Octave is the computational/visualization engine used by UMSEE considering it is free and does not present any licensing issues. Cellular phone messaging (i.e. SMS and MMS) allows easy exchange of multimedia data, and is very popular amongst the youth and thus is a great segue to interest students in STEM education. Social networking technology are free to use, used regularly by youth, and is a socially relevant way of engaging students.

UMSEE is a system that is proposed to merge SMS/MMS messaging and scientific computational/visualization engines. The initial design of UMSEE used a dedicated email as the interface between SMS/MMS messaging and Octave but it proved to be very cumbersome arising from the fact that it cannot handle SMS/MMS messages in a standard manner. The social networking website Twitter, is an excellent alternative to using a dedicated email account because of its inherent SMS capabilities. Moreover the Twitter based system offers better versatility by allowing input from users through SMS/MMS messaging as well as web based interface. MATLAB Mobile, an iPhone application that allows pervasive use of MATLAB, is direct competition to the UMSEE system but has shortcomings. The most notable shortcomings are its inability to create M-files, and the need to have a copy of MATLAB installed on a computer or desktop. The experiments in Tables 3 and 4 proved that the Twitter interface is the interface of choice considering the success rate results (100% versus 53%).

5.2 Contributions

The UMSEE system will be a cornerstone for future research in the area of STEM educational enhancement research within the PSEENCAT group. It provides students in the PSEE (Pervasive Systems of Electrical Engineering) group a system interface for future projects. Futhermore, this system is a novel paradigm in the field of STEM educational enhancement. UMSEE potentially sets a standard in pervasive computing and education by giving educators an amendable platform to work with. Imagine a classroom where lessons are taught via text messaging and picture messages instead of blackboards and note taking. This allows the days interactive lesson to be easily continued after the class has been dismissed, even while the students and instructors are at home.

5.3 Recommendations for Future Direction

The first improvement to be made should be the ability to handle audio and video inputs. The ability to handle audio and video would complete the UMSEE system, in the sense of its multimedia handling capabilities. Additionally, future researchers of UMSEE can possibly investigate the use of other social networking SMS/MMS interfaces, or other third party applications that can possibly handle audio and video such as Twaudio, and Cynchcast. Future researchers can explore using other social networking sites besides Twitter, (e.g. Facebook, Youtube). Next would be to simplify the process of sending information. As seen in Table 3, there are 14 characters for the message overhead when creating M-files, and at least 17 characters for the message overhead when sending images. Also, there may be an alternative to using Twitter accounts, which may shorten the initial steps needed to use the UMSEE system. Throughout the development of UMSEE, the a major issue was alerting the user of Octave syntax errors when he/she creates M-files. Finding a way to seamlessly notify the user of syntax errors would speed system processing, reduce erro33r, and minimize the amount of SMS/MMS traffic to and from the user.

REFERENCES

Bondi, H. (2002). *Preparation for careers in engineering and science*, *135*(4), 227.

Chapman, S. (2010, November 8). *The History of internet search and google* [Video file]. Retrieved from http://www.zdnet.com/blog/seo

Eisenberg, M. (2007). Pervasive fabrication: Making construction ubiquitous in education (p. 193). Presented at the PerCom Workshops '07. Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, White Plains, NY. doi:10.1109/PERCOMW.2007.93

Fehrenbacher, K. (2007, March 16). 10 *Things to know about short codes*. Retrieved from http://gigaom.com/2007/03/16/10-things-to-know-about-short-codes/

Firdaus bin Haji Sidek, S. (2010). The Development of the short messaging service (SMS) application for the school usage (Vol. 3, pp. 1382 - 1386). Presented at the 2010 International Symposium in Information Technology, Kuala Lumpur. doi:10.1109/ITSIM.2010.5561647

Fontenot, D., Chandler, J., Talkmitt, S., & Sullivan, K. (2008). The Texas high school initiative aims at STEM education reform: Texas tech university T-STEM center—putting the “E” in K-12 STEM education (pp. F2B-1). Presented at the FIE '07. 37th Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports, Milwaukee, WI. doi:10.1109/FIE.2007.4418096

Gable, L., & Recio, R. (2009). How industry can Help Improve STEM Graduation Rates (Vol. 50, p. 1). Presented at the 2007 IEEE Meeting the Growing Demand for Engineers and Their Educators 2010-2020 International Summit, Munich. doi:10.1109/MGDETE.2007.4760357

Gehrig, G. B., Abrams, L., Bosley, D., Conrad, J., & Kuyath, S. (2009). Addressing the demand for engineers by teaching engineering to counselors and teachers (Vol. 50, p. 1). Presented at the 2007 IEEE Meeting the Growing Demand for Engineers and Their Educators 2010-2020 International Summit, Munich. doi:10.1109/MGDETE.2007.4760353

GNU Octave Documentation. (n.d.). . GNU. Retrieved from http://www.gnu.org/software/octave/do

Honeycutt, C., & Herring, S. C. (2009). Beyond microblogging: Conversation and collaboration via Twitter (p. 1). Presented at the 42nd Hawaii International Conference on System Sciences, Big Island, HI. doi:10.1109/HICSS.2009.89

How to find your Twitter short code or long code. (n.d.). *Twitter Help Center*. Retrieved from http://support.twitter.com/entries/14226-how-to-find-your-twitter-short-long-code

Huang, S., & Mangs, J. (2008). Pervasive computing: Migrating to mobile devices: a case study (p. 1). Presented at the 2nd Annual IEEE Systems Conference,, Montreal, Que. doi:10.1109/SYSTEMS.2008.4519056

Kiyohara, S. (2009). A Study on how technological innovation affected the 2008 U.S. presidential election: Young voters' participation and Obama's victory (p. 223). Presented at the SAINT '09. Ninth Annual International Symposium on Applications and the Internet, Bellevue, WA. doi:10.1109/SAINT.2009.51

Kramer, B., & Strohlein, G. (2006). Exploring the use of cellular phones for pervasive eLearning (p. 6 pp). Presented at the PerCom Workshops 2006. Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, Pisa. doi:10.1109/PERCOMW.2006.54

Leena Rao. (2009). The *Top 21 Twitter applications (according to compete)*. Retrieved from http://techcrunch.com/2009/02/19/the-top-20-twitter-applications/

Lenhart, A., Ling, R., Campbell, S., & Purcell, K. (2010). *Teens and mobile phones*. Washington, D.C.: Pew Research Center's Internet & American Life Project. Retrieved from http://pewinternet.org/~/media//Files/Reports/2010/PIP-Teens-and-Mobile-2010-with-topline.pdf

Magill, J., & Roy, S. (2010). *Chips for everyone: A multifaceted approach in electrical engineering outreach*, *53*(1), 114. doi:10.1109/TE.2009.2025267

Mahmood, H. B. (2003). Transport layer security protocol in telnet (Vol. 3, p. 1033). Presented at the The 9th Asia-Pacific Conference on Communications. doi:10.1109/APCC.2003.1274255

MATLAB Mobile. (n.d.). . MathWorks. Retrieved from http://www.mathworks.com/mobile/

Moler, C. (n.d.). *Experiments with MATLAB* (Electronic.). MathWorks. Retrieved from http://www.mathworks.com/moler/exm/index.html

Mostafa., M. E. (2002). MMS - The Modern wireless solution for multimedia messaging (Vol. 5, p. 2466). Presented at the The 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications. doi:10.1109/PIMRC.2002.1046587

Nations, D. (2010). *The Top 10 most popular social networks*. Retrieved from http://webtrends.about.com/b/2010/03/15/the-top-10-most-popular-social-networks.htm

Oauth. (2010, September 30). . *Hueniverse*. Retrieved from http://hueniverse.com/oauth/

Pulecio, J. F., Pulecio, A., Westlake, M., & Bhanja, S. (2010). A Snapshot of young America's perspective towards STEM (pp. S2E-1). Presented at the IEEE Frontiers in Education Conference (FIE), Washington, D.C. doi:10.1109/FIE.2010.5673221

Reddy, Y. (2006). Pervasive computing: Implications, opportunities and challenges for the society (p. 5). Presented at the 1st International Symposium on Pervasive Computing and Applications, Urumqi. doi:10.1109/SPCA.2006.297455

Science and engineering indicators 2010 (Biennial Science Indicators No. 19). (2010). . National Science Board.

Shibu, S., & Jain, S. (2010). Pervasive computing for automobiles: An approach to maximize user convenience and safety using VANETs. *International Journal of Computer and Electrical Engineering*, *2*(6).

Solis, B. (2009, October 26). *Teens adopting Twitter*. *www.briansolis.com*. Retrieved from http://www.briansolis.com/2009/10/the-youth-are-taking-over-twitter/

St. George, D. (2010). *Twitter not all that popular among teenagers*. Washington, D.C.: Washington Post. Retrieved from http://www.washingtonpost.com/wp-dyn/content/article/2010/02/03/AR2010020302591.html

Twitter API Wiki. (n.d.). . Retrieved from http://apiwiki.twitter.com/w/page/22554648/FrontPage

Whitman, L. E., & Witherspoon, T. L. (2004). Using legos to interest high school students and improve k12 STEM education (Vol. 2, pp. F3A_6 - F3A_10). Presented at the FIE 2003. 33rd Annual Frontiers in Education, Boulder, CO. doi:10.1109/FIE.2003.1264721

APPENDIX A.   TWITTER BASED APPLICATION

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Web;
using System.Xml;
using System.IO;
using System.Diagnostics;
using System.Runtime.Serialization.Json;
using System.Text.RegularExpressions;
using System.Net.Sockets;
using System.Net.Security;
using System.Net.Mail;

namespace CodingTheTweet
{
    public partial class MatLabTwitterApp : Form
    {

       //GraphingTweet contains the function to be graphed
       // string GraphingTweet = "";

        //This string contains the standard graphing
          //  syntax needed to graph an equation in Matlab
       // string StandardGraphingSyntax = "function [ ret ] = graph
()\n" +
         //                              "x = [-100:100];\n";
      //  string StandardSavingSyntax = "print -djpg graph.jpg;\n" +
                          //            "close;\n";
        string p, h, k = "";

        //object to invoke the cloudvox web service
        private WebClient client = new WebClient();

        //Cell Number to email result
        string Number = "";
        string Carrier = "";
        string SentFrom = "";
        string TwitpicUrl = "";
        string TwitpicCommand = "";
        string LastMention = "";
        string CurrentTweet = "";
        string usersfile = "";
        string mfile = "";
        string PreviousMention = "";
        Tweets TweetObj = new Tweets();

        public MatLabTwitterApp()
        {
            InitializeComponent();
           //add DownloadStringCompleted event handler to WebClient


           client.DownloadStringCompleted += new
DownloadStringCompletedEventHandler(client_DownloadStringCompleted);


        }//end constructor


        private void client_DownloadStringCompleted(object sender,
DownloadStringCompletedEventArgs e)
        {
           // MessageBox.Show("Getting Carrier: " + e.ToString());
            //check if any error occurred in retrieving service data
            if (e.Error == null)
            {
                string JSONstring = e.Result;
               // MessageBox.Show("Extracting Data");
                //display Number To Send Data
                Carrier =
JSONstring.Substring(JSONstring.IndexOf("carrier_name") + 15,
JSONstring.IndexOf("effective_on") - JSONstring.IndexOf("carrier_name")
- 18).Trim();
                if (JSONstring.Contains("wireless")) //is it a wireless
num?
                {
                    if (Carrier.Contains("SPRINT"))//if carrier is
sprint
                        Number = string.Concat(Number,
"@pm.sprint.com");
                    else if (Carrier.Contains("VERIZON"))//if carrier
is sprint
                        Number = string.Concat(Number, "@vzwpix.com");
                    else if (Carrier.Contains("ALLTELL"))//if carrier
is sprint
                        Number = string.Concat(Number,
"@mms.alltel.net");
                    else if (Carrier.Contains("ALLTELL"))//if carrier
is sprint
                        Number = string.Concat(Number,
"@mms.alltel.net");
                    else if (Carrier.Contains("ATT"))//if carrier is
sprint
                        Number = string.Concat(Number, "@mms.att.net");
                    else if (Carrier.Contains("CRICKET"))//if carrier
is sprint
```

```
                        Number = string.Concat(Number,
"@mms.mycricket.com");
                    else if (Carrier.Contains("NEXTEL"))//if carrier is
sprint
                        Number = string.Concat(Number,
"@messaging.nextel.com");
                    else if (Carrier.Contains("ALLTELL"))//if carrier
is sprint
                        Number = string.Concat(Number,
"@mms.alltel.net");

                    //MessageBox.Show(Number, "Number To Send Data");
                   // TweetObj.Email(Number, GraphingTweet);
                    TweetCheck.Enabled = true;

                }

                else
                    MessageBox.Show("Not a wireless phone", "Info");
                //break;

            }//end if
            else
                MessageBox.Show("There was a problem with the cloudvox
service" + e.Error.ToString());

        }//end WebClient Event


        private void SendTweet()
        {
            if (txtTweet.Text.Length == 0)
            {
                MessageBox.Show("Your tweet must be at least 1
character long!");
                return;
            }

            try
            {
                // URL-encode the tweet...
                string tweet = HttpUtility.UrlEncode(txtTweet.Text);

                // And send it off...
                string xml = _oAuth.oAuthWebRequest(
                    oAuthTwitter.Method.POST,
                    "http://twitter.com/statuses/update.xml",
                    "status="+tweet);

            }


            catch (Exception ex)
            {
                MessageBox.Show("An error occurred while posting your
tweet:\n\n" + ex.Message);
                return;
            }

            txtTweet.Text = String.Empty;
        }//end SendTweet

        private void SendDM(string SendTo, string Message)
        {
            string SendDM = "";
            SendDM = _oAuth.oAuthWebRequest(
                                 oAuthTwitter.Method.POST,

"http://api.twitter.com/1/direct_messages/new.xml", "?screen_name=" +
SendTo + "&text=" + Message);

        }//end SendDM

        private void DestroyDM(string TweetID)
        {
            string DestroyedID = "";
            DestroyedID = _oAuth.oAuthWebRequest(
             oAuthTwitter.Method.POST,
             "http://api.twitter.com/1/direct_messages/destroy/" +
TweetID + ".xml",
             "screen_name=PSEENCAT");
        }//end DestroyDM

        public void SendMessage(string SendTo, string MessageToSend)
        {
            try
            {
                //create the mail message
                MailMessage mail = new MailMessage();
                //mail.From = new MailAddress("mychatlab@gmail.com");
                mail.From = new MailAddress("bcjudd@ncat.edu");
                mail.To.Add(SendTo);
                mail.Body = MessageToSend;
                //SmtpClient smtp = new SmtpClient("smtp.gmail.com");
                SmtpClient smtp = new SmtpClient("smtp.ncat.edu");
                //smtp.EnableSsl = true;
                //smtp.Credentials = new
NetworkCredential("mychatlab@gmail.com", "Verysimple");

                smtp.Send(mail);
            }//end try
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString(), "SendMessage");
            }

        }//end SendMessage
        private void btwTweet_Click(object sender, EventArgs e)
        {
            SendTweet();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            txtTweet.Enabled = btwTweet.Enabled = IsConfigured;

            // Set up our credentials...
            _oAuth.Token = Settings1.Default.token;
            _oAuth.TokenSecret = Settings1.Default.secretToken;
            _oAuth.ConsumerKey = Settings1.Default.consumerKey;
            _oAuth.ConsumerSecret = Settings1.Default.consumerSecret;
            _oAuth.PIN = Settings1.Default.pin;

        }

        private void txtTweet_TextChanged(object sender, EventArgs e)
        {
            lblCount.Text = (140 - txtTweet.Text.Length).ToString();
        }

        private void txtTweet_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Return)
                SendTweet();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            TweetCheck.Enabled = false;
            FormSettings dlgSettings = new FormSettings(_oAuth);
            if (DialogResult.OK == dlgSettings.ShowDialog())
            {
                txtTweet.Enabled = btwTweet.Enabled = IsConfigured;
                TweetCheck.Enabled = true;
            }

        }

        public bool IsConfigured
        {
            get
            {
                return !String.IsNullOrEmpty(Settings1.Default.token)
&&

!String.IsNullOrEmpty(Settings1.Default.secretToken) &&

!String.IsNullOrEmpty(Settings1.Default.consumerKey) &&
!String.IsNullOrEmpty(Settings1.Default.consumerSecret) &&
                    !String.IsNullOrEmpty(Settings1.Default.pin);
                 }
        }

        private oAuthTwitter _oAuth = new oAuthTwitter();

        private void ShowTweets_Click(object sender, EventArgs e)
        {
            TweetCheck.Enabled = true;
        }//end ShowTweets_Click

        private void TweetCheck_Tick(object sender, EventArgs e)
        {
            StreamWriter sw;
            StreamWriter sw2;

            //Create the Database if it doesn't exist
            if (!Directory.Exists("c:\\UMSEE"))
                Directory.CreateDirectory("c:\\UMSEE");
            if (!File.Exists("c:\\UMSEE\\usersfile.txt"))
            {
                sw = new StreamWriter("c:\\UMSEE\\usersfile.txt");
                sw.Close();
            }//end if
            if(!File.Exists("c:\\UMSEE\\ProgramInfo"))
            {
                sw2 = new StreamWriter("c:\\UMSEE\\ProgramInfo");
                sw2.WriteLine(LastMention);
                sw2.Close();
            }//end if

            //Read the last mention
            LastMention =
File.ReadAllText("c:\\UMSEE\\ProgramInfo").Trim();

            //**************Parse the XML page and display the last 20
tweets**************

            /*initialize a new instance of the XmlDocument class*/
            XmlDocument xDoc = new XmlDocument();
            XmlDocument xDoc2 = new XmlDocument();
            string DirectMessage = "";
            string Mentions = "";
            string ScreenNameAndNumber = "";

            //load the XML document from the twitter stream
           //try
           // {
                DirectMessage = _oAuth.oAuthWebRequest(
                        oAuthTwitter.Method.GET,
                        "http://api.twitter.com/1/direct_messages.xml",
                        "screen_name=PSEENCAT");
                xDoc.InnerXml = DirectMessage;
                if (LastMention == "")
                {
                    //check the mentions as well
                    Mentions = _oAuth.oAuthWebRequest(
                        oAuthTwitter.Method.GET,

"http://api.twitter.com/1/statuses/mentions.xml",
                        "screen_name=PSEENCAT");
                    xDoc2.InnerXml = Mentions;
                }//end if
                else
                {
                    Mentions = _oAuth.oAuthWebRequest(
                        oAuthTwitter.Method.GET,

"http://api.twitter.com/1/statuses/mentions.xml",
                        "screen_name=PSEENCAT&sinceid=" +
LastMention);
                    xDoc2.InnerXml = Mentions;
                }//end else

                //method GetElementsByTagName() obtains the addresses
of a collection of elements that match the specified name
                XmlNodeList tweets = xDoc.GetElementsByTagName("text");
                XmlNodeList tweetid = xDoc.GetElementsByTagName("id");
                XmlNodeList screen_name =
xDoc.GetElementsByTagName("sender_screen_name");
                //Mentions
                XmlNodeList Mentionid =
xDoc2.GetElementsByTagName("id");
                XmlNodeList MentionTweet =
xDoc2.GetElementsByTagName("text");
                XmlNodeList Mentionscreen_name =
xDoc2.GetElementsByTagName("screen_name");

                //Create an array to convert the XmlNodeList to an
array
                string[] tweetarray = new string[tweets.Count];
                string[] tweetidarray = new string[tweetid.Count];
                string[] screen_namearray = new
string[screen_name.Count];
                //mentions
                string[] Mentionidarray = new string[Mentionid.Count];
                string[] Mentionscreen_namearray = new
string[Mentionscreen_name.Count];
                string[] MentionTweetarray = new
string[MentionTweet.Count];

                PreviousTweets.Items.Clear();
                //Populate MentionTweet
                for (int i = 0; i < MentionTweet.Count; i++)
                {
                    Mentionidarray[i] =
Mentionid[i].InnerText.ToString();
                    Mentionscreen_namearray[i] =
Mentionscreen_name[i].InnerText.ToString();
                    MentionTweetarray[i] =
MentionTweet[i].InnerText.ToString();
                }
                //populate DMs
                for (int i = 0; i < tweets.Count; i++)
                {
                    //store all of the XmlNodeList into the array
                    tweetarray[i] = tweets[i].InnerText.ToString();
                    tweetidarray[i] = tweetid[i].InnerText.ToString();
                    screen_namearray[i] =
screen_name[i].InnerText.ToString();
                    //mentions

                    //Populate the ListBox with the last 20 tweets
                    PreviousTweets.Items.Add(tweets[i].InnerXml);
                }

                //Search mentions for twitpics
                bool FoundParab = false;
                bool FoundTwitpic = false;
                bool FoundMFile = false;
                bool FoundTwitpicCommand = false;
                for (int i = 0; i < MentionTweetarray.Count(); i++)
                {
                    if (LastMention == Mentionidarray[i])
                    {
                       // MessageBox.Show("No new Mention");
                        goto skipMentions;
                    }//end if
                    if(MentionTweetarray[i].Contains("twitpic"))
                    {
                        TweetCheck.Enabled = false;  // disable timer
                        //save the id
                        PreviousMention = LastMention;
                        LastMention = Mentionidarray[i];
                        sw2 = new
StreamWriter("c:\\UMSEE\\ProgramInfo");
                        sw2.WriteLine(LastMention);
                        sw2.Close();
                        MentionTweetarray[i] =
MentionTweetarray[i].ToLower();
                        if(MentionTweetarray[i].Contains("??"))
                        {
                            FoundTwitpicCommand = true;
                            TwitpicCommand =
MentionTweetarray[i].Substring(MentionTweetarray[i].IndexOf("??")+2);
                            TwitpicCommand =
TwitpicCommand.Substring(0, TwitpicCommand.IndexOf("??"));
                        }
                        string[] words = MentionTweetarray[i].Split('
');
                        foreach (string word in words)
                        {
                             if (word.Contains("twitpic"))
                            {
                                TwitpicUrl = word.Trim();
                                break;
                            }
                        }//end foreach

                       // MessageBox.Show("twitpic Url " + TwitpicUrl,
"Twitpic Url");
                        usersfile =
System.IO.File.ReadAllText("c:\\UMSEE\\usersfile.txt");
                        if
(!usersfile.Contains(Mentionscreen_namearray[i]))
                        {
                           // MessageBox.Show("User not found in
Database " + Mentionscreen_namearray[i], "User Not Found");
                            string register_message = "No number found,
user must register and resend request...";
                            SendDM(Mentionscreen_namearray[i],
register_message);
                                              goto skipMentions;
                        }//end if
                        else
                        {
                            FoundTwitpic = true;
                            SentFrom = Mentionscreen_namearray[i];
                            int index =
usersfile.IndexOf(Mentionscreen_namearray[i]) +
Mentionscreen_namearray[i].Length;
                            Number = usersfile.Substring(index,
usersfile.IndexOf(';',usersfile.IndexOf(SentFrom)) - index).Trim();
                          //  MessageBox.Show("Number To Send to " +
Number, "Number");

                            goto GetTwitpic;

                        }//end else
                    }//end if
                }//end for

                //search tweetarray and return the index of the
graphing tweet

skipMentions:
                //Go through every tweet
                for (int i = 0; i < tweetarray.Count(); i++)
                {
                    bool contains_reg = tweetarray[i].IndexOf("reg",
StringComparison.OrdinalIgnoreCase) >= 0;
                    //if (tweetarray[i].Contains("reg"))
                    if (contains_reg)
                    {
                        MessageBox.Show("Registering: " +
screen_namearray[i]);
                        TweetCheck.Enabled = false;
                        //Number = tweetarray[i];
                        string[] vals = tweetarray[i].Split(' ');
                        Number = vals[1];
                        Carrier = vals[2];
                       // Number =
Number.Substring(Number.IndexOf("cell") + 4).Trim();
                       // Number = Number.Replace(" ", "");
                        //Carrier =
Number.Substring(10).Trim().ToLower();
                        //Number = Number.Substring(0, 10);

                        if (!Regex.IsMatch(Number, @"^\d{10}$"))
                        {
                            string CellError_message = "Cell phone
number should be a 10 digit number with no spaces, aphabets or
hyphens";
                            SendDM(screen_namearray[i],
CellError_message);
                            DestroyDM(tweetidarray[i]);
                            break;
                        }//end if

                        if (Carrier.Contains("sprint"))//if carrier is
sprint
                            Number = string.Concat(Number,
"@pm.sprint.com");
                        else if (Carrier.Contains("verizon"))//if
carrier is sprint
                            Number = string.Concat(Number,
"@vzwpix.com");
                        else if (Carrier.Contains("alltell"))//if
carrier is sprint
                            Number = string.Concat(Number,
"@mms.alltel.net");
                        else if (Carrier.Contains("alltell"))//if
carrier is sprint
                            Number = string.Concat(Number,
"@mms.alltel.net");
                        else if (Carrier.Contains("att"))//if carrier
is sprint
                            Number = string.Concat(Number,
"@mms.att.net");
                        else if (Carrier.Contains("cricket"))//if
carrier is sprint
                            Number = string.Concat(Number,
"@mms.mycricket.com");
                        else if (Carrier.Contains("nextel"))//if
carrier is sprint
```

```
                            Number = string.Concat(Number, 
"@messaging.nextel.com");
                        else if (Carrier.Contains("alltell"))//if 
carrier is sprint
                            Number = string.Concat(Number, 
"@mms.alltel.net");
                        else if (Carrier.Contains("tmobile"))//if 
carrier is sprint
                            Number = string.Concat(Number, 
"@tmomail.net");
                        else
                        {
                          //  MessageBox.Show("Registration DM doesn't 
contain correct carrier", "Error!");

                            string CarrierError_message = "Please use 
valid carrier names: "
                                    + "sprint, verizon, alltell, att, 
cricket, nextel, alltell, tmobile";


                            SendDM(screen_namearray[i], 
CarrierError_message);
                            DestroyDM(tweetidarray[i]);

                            break;
                        }//end else

                        ScreenNameAndNumber = screen_namearray[i] + " " 
+ Number;

                        usersfile = 
System.IO.File.ReadAllText("c:\\UMSEE\\usersfile.txt");
                        if (!usersfile.Contains(screen_namearray[i]))
                        {
                            MessageBox.Show("Writing To File: " + 
ScreenNameAndNumber);
                            //store the screen name and number
                            using (StreamWriter sw1 = new 
StreamWriter("c:\\UMSEE\\usersfile.txt", true))
                            {
                                sw1.WriteLine(ScreenNameAndNumber + 
";\n");
                                sw1.Close();
                            }//end using
                            //Delete the requested direct message
                            MessageBox.Show("Wrote To File, Now 
Distroying Message");
                            DestroyDM(tweetidarray[i]);

                            //   MessageBox.Show("Registered user " + 
screen_namearray[i], "Registration");
                            string SuccessfulRegistration_message = 
"You have been successfully registered, please double"
```

```
                                + "check info. Name: " +
screen_namearray[i] + " Number: " + Number;
                            SendMessage(Number,
SuccessfulRegistration_message);
                        }//end if
                        else
                        {
                            string numberToText = "";
                            string nameToSend = screen_namearray[i];
                            numberToText =
usersfile.Substring(usersfile.IndexOf(nameToSend) +

nameToSend.Length);
                            numberToText = numberToText.Substring(0,
numberToText.IndexOf(';')).Trim();
                            string AlreadyRegistered_message = "You
have already registered with number: " + numberToText;
                            MessageBox.Show("User " + nameToSend + "
already registered with number: " + numberToText);
                            DestroyDM(tweetidarray[i]);
                            SendMessage(numberToText,
AlreadyRegistered_message);
                        }//end else
                    }//end if

                    //dm is a requeset

                    else if (tweetarray[i].Contains("mfile"))
                    {
                        TweetCheck.Enabled = false;  // disable timer
                        usersfile =
System.IO.File.ReadAllText("c:\\UMSEE\\usersfile.txt");
                        mfile =
tweetarray[i].Substring(tweetarray[i].IndexOf("mfile") + 5);

                        //MessageBox.Show("Checking database...");
                        if (!usersfile.Contains(screen_namearray[i]))
                        {
                         //   MessageBox.Show("User not found in
Database " + screen_namearray[i], "User Not Found");
                            // string reg_message =
HttpUtility.UrlEncode("No number found, user must register and resend
request...");
                            string register_message = "User not
registered. To register text reg num (no spaces or hyphens)" +
                                                          "carrier. Ex: reg
8003334444 att";

                            SendDM(screen_namearray[i],
register_message);
                            //Delete the requested direct message
                            DestroyDM(tweetidarray[i]);

                        }//end if
```

```
                            else
                            {
                                FoundMFile = true;
                                SentFrom = screen_namearray[i];
                                int index = 
usersfile.IndexOf(screen_namearray[i]) + screen_namearray[i].Length;
                                Number = usersfile.Substring(index, 
usersfile.IndexOf(';', usersfile.IndexOf(SentFrom)) - index).Trim();
                               // MessageBox.Show("M-File Found!", "M-File 
Found");
                                //Delete the requested direct message
                                DestroyDM(tweetidarray[i]);

                            }//end else
                            break;
                        }//end else if

                        else if (tweetarray[i].Contains("parab"))
                        {
                            TweetCheck.Enabled = false;  // disable timer

                            usersfile = 
System.IO.File.ReadAllText("c:\\UMSEE\\usersfile.txt");
                            tweetarray[i] = 
tweetarray[i].Substring(tweetarray[i].IndexOf("parab"));
                            string[] ParabValues = tweetarray[i].Split('
');


                                p = ParabValues[1];
                                h = ParabValues[2];
                                k = ParabValues[3];
                                double Num;
                                bool isNum = double.TryParse(p + h + k, out 
Num);



                          //  MessageBox.Show("Checking database...");
                            if (!usersfile.Contains(screen_namearray[i]))
                            {
                                string register_message = "User not 
registered. To register text reg num (no spaces or hyphens)" +
                                                                  "carrier. Ex: reg 
8003334444 att";
                                SendDM(screen_namearray[i], 
register_message);

                                //Delete the requested direct message
                                DestroyDM(tweetidarray[i]);

                            }//end if
                        else if (!isNum)
                        {
                            string Invalid_Argument = "The values for
p, h, and k must be numeric characters." +
                                                          " Send your
request again in the correct format:" +
                                                          "parab p h k
(case doesn't matter)";
                            Number =
usersfile.Substring(usersfile.IndexOf(screen_namearray[i]) +
screen_namearray[i].Length);
                            Number =
Number.Substring(0,Number.IndexOf(';')).Trim();
                            MessageBox.Show(Number);
                            SendMessage(Number, Invalid_Argument);
                            //Delete the requested direct message
                            DestroyDM(tweetidarray[i]);
                            CurrentTweet = tweetidarray[i];
                            break;
                        }//end else if
                        else
                        {
                            FoundParab = true;
                            SentFrom = screen_namearray[i];
                            int index =
usersfile.IndexOf(screen_namearray[i]) + screen_namearray[i].Length;
                            Number = usersfile.Substring(index,
usersfile.IndexOf(';', usersfile.IndexOf(SentFrom)) - index).Trim();
                            //Delete the requested direct message
                            DestroyDM(tweetidarray[i]);
                        }//end else

                        break;
                    }//end else if

                }//end for

                // write a line of text to the file
            GetTwitpic:
                if (FoundParab)
                {
                    string path = "c:\\UMSEE\\Text";
                    if (!Directory.Exists(path))
                        Directory.CreateDirectory(path);
                    string Timestamp = "\\A" +
DateTime.Now.DayOfYear.ToString() +
DateTime.Now.TimeOfDay.ToString().Substring(0, 5).Replace(":", "");

                    //Create the dot M and write the equation to it
                    TextWriter tw = new StreamWriter(path + "\\" +
Timestamp + ".txt");

                    // tw.Write(tweetarray[retVal]);
                    tw.WriteLine("## " + SentFrom);
                    tw.WriteLine("## " + Number);
                    tw.WriteLine("parab(" + p + ',' + h + ',' + k +
")");
                    // close the stream
                    tw.Close();
                   // MessageBox.Show("Text file has been created");
                }//end if

                else if (FoundMFile)
                {
                    string path = "c:\\UMSEE\\MFile";
                    if (!Directory.Exists(path))
                        Directory.CreateDirectory(path);

                   /* //Create Folder Text Folder for the user
                    string UserFolder = "c:\\UMSEE\\MFile\\" +
SentFrom;
                    if (!Directory.Exists(UserFolder))
                        Directory.CreateDirectory(UserFolder);*/

                    //Create the dot M and write the info to it
                    string Timestamp = "\\A" +
DateTime.Now.DayOfYear.ToString() +
DateTime.Now.TimeOfDay.ToString().Substring(0, 5).Replace(":", "");

                    TextWriter tw = new StreamWriter(path + Timestamp +
".m");

                    // tw.Write(tweetarray[retVal]);
                    tw.WriteLine("## " + SentFrom + ';');
                    tw.WriteLine("## " + Number + ';');
                    tw.WriteLine(mfile);
                    // close the stream
                    tw.Close();
                   // MessageBox.Show("M file has been created");`

                }//end else if

                else if (FoundTwitpic)
                {
                    string path = "c:\\UMSEE\\Image";
                    if (!Directory.Exists(path))
                        Directory.CreateDirectory(path);
                    string timestamp = "\\A" +
DateTime.Now.DayOfYear.ToString() +
DateTime.Now.TimeOfDay.ToString().Substring(0, 5).Replace(":", "");

                    if (FoundTwitpicCommand)
                    {
                        TextWriter tw1 = new StreamWriter(path + "\\" +
timestamp + ".m");

                        // tw.Write(tweetarray[retVal]);
                        tw1.WriteLine("## " + SentFrom + ';');
```

```
                                    tw1.WriteLine("## " + Number + ';');
                                    tw1.WriteLine(TwitpicCommand);
                                    // close the stream
                                    tw1.Close();
                                }//end if
                                else
                                {
                                    TextWriter tw1 = new StreamWriter(path + "\\" +
timestamp + ".m");

                                    // tw.Write(tweetarray[retVal]);
                                    tw1.WriteLine("## " + SentFrom + ';');
                                    tw1.WriteLine("## " + Number + ';');
                                    // close the stream
                                    tw1.Close();

                                }//end else

                                //Create Folder Text Folder for the user

                                HttpWebRequest request = (HttpWebRequest)
                                WebRequest.Create(TwitpicUrl);
                                // execute the request
                                HttpWebResponse response = (HttpWebResponse)
                                request.GetResponse();
                                // we will read data via the response stream
                                Stream ReceiveStream =
response.GetResponseStream();
                                StreamReader readStream = new
StreamReader(ReceiveStream);
                                string contents = readStream.ReadToEnd();
                                contents =
contents.Substring(contents.IndexOf("photo-display") + 20);
                                contents = contents.Substring(0,
contents.IndexOf("alt=") - 2);
                                string imageUrl = contents;
                               // MessageBox.Show(imageUrl, "Image URL");
                                string saveLocation = path + "\\" + timestamp
                                                      + ".jpg";
                                MessageBox.Show("Saving image...");
                                byte[] imageBytes;
                                HttpWebRequest imageRequest =
(HttpWebRequest)WebRequest.Create(imageUrl);
                                WebResponse imageResponse =
imageRequest.GetResponse();

                                Stream responseStream =
imageResponse.GetResponseStream();

                                using (BinaryReader br = new
BinaryReader(responseStream))
                                {
                                    imageBytes = br.ReadBytes(500000);
                                    br.Close();
```

```
                    }

                    responseStream.Close();
                    imageResponse.Close();

                    FileStream fs = new FileStream(saveLocation,
FileMode.Create);
                    BinaryWriter bw = new BinaryWriter(fs);
                    try
                    {
                        bw.Write(imageBytes);
                    }
                    finally
                    {
                        fs.Close();
                        bw.Close();
                    }


                }//end else if

                TweetCheck.Enabled = true;

          // }//end try
         /*   catch (Exception exceptionParameter)
            {
                //Twitter Error Codes
                string message = exceptionParameter.Message;
                if (message.Contains("304"))
                    MessageBox.Show("There was no new data to return",
" 304 Not Modified");
                else if (message.Contains("400 "))
                    MessageBox.Show("The request was invalid, possibly
rate limited", "400 Bad Request");
                else if (message.Contains("401"))
                    MessageBox.Show("Authentication credentials were
missing or incorrect", "401 Unauthorized");
                else if (message.Contains("403"))
                    MessageBox.Show("The request is understood, but it
has been refused," +
                                    " possibly denied due to upate
limits", "403 Forbidden");
                else if (message.Contains("404 "))
                    MessageBox.Show("The URI requested is invalid or
the resource requested," +
                                    " such as a user, does not exists",
"404 Not Found");
                else if (message.Contains("406"))
                    MessageBox.Show("Returned by the Search API when an
invalid format is" +
                                    " specified in the reques", "406
Not Acceptable");
                else if (message.Contains("420"))
```

```
                    MessageBox.Show("Returned by the Search and Trends
API, you are being rate limited", "420 Enhance Your Calm");
                else if (message.Contains("500"))
                    MessageBox.Show("Something is broken, please report
to Twitter", "500 Internal Server Error");
                else if (message.Contains("502"))
                    MessageBox.Show("Twitter is down or being
upgraded", "502 Bad Gateway");
                else if (message.Contains("503"))
                    MessageBox.Show("The Twitter servers are up, but
overloaded with requests. Try again later"
                                    , "503 Service Unavailable");
                //General Message
                else
                    MessageBox.Show(exceptionParameter.Message,
"General Message");
                TweetCheck.Enabled = true;
            }//end catch*/

        } //end TweetCheck_Tick



    }//end class Form1

}//end namespace

using System;
using System.Security.Cryptography;
using System.Collections.Generic;
using System.Text;
using System.Web;

namespace CodingTheTweet
{
    public class OAuthBase
    {

        /// <summary>
        /// Provides a predefined set of algorithms that are supported
officially by the protocol
        /// </summary>
        public enum SignatureTypes
        {
            HMACSHA1,
            PLAINTEXT,
            RSASHA1
        }

        /// <summary>
        /// Provides an internal structure to sort the query parameter
        /// </summary>
        protected class QueryParameter
        {
            private string name = null;
            private string value = null;

            public QueryParameter(string name, string value)
            {
                this.name = name;
                this.value = value;
            }

            public string Name
            {
                get { return name; }
            }

            public string Value
            {
                get { return value; }
            }
        }

        /// <summary>
        /// Comparer class used to perform the sorting of the query
parameters
        /// </summary>
        protected class QueryParameterComparer :
IComparer<QueryParameter>
        {

            #region IComparer<QueryParameter> Members

            public int Compare(QueryParameter x, QueryParameter y)
            {
                if (x.Name == y.Name)
                {
                    return string.Compare(x.Value, y.Value);
                }
                else
                {
                    return string.Compare(x.Name, y.Name);
                }
            }

            #endregion
        }

        protected const string OAuthVersion = "1.0";
        protected const string OAuthParameterPrefix = "oauth_";

        //
        // List of know and used oauth parameters' names
        //
        protected const string OAuthConsumerKeyKey =
"oauth_consumer_key";
```

```
        protected const string OAuthCallbackKey = "oauth_callback";
        protected const string OAuthVersionKey = "oauth_version";
        protected const string OAuthSignatureMethodKey =
"oauth_signature_method";
        protected const string OAuthSignatureKey = "oauth_signature";
        protected const string OAuthTimestampKey = "oauth_timestamp";
        protected const string OAuthNonceKey = "oauth_nonce";
        protected const string OAuthTokenKey = "oauth_token";
        protected const string OAuthTokenSecretKey =
"oauth_token_secret";
        protected const string OAuthVerifierKey = "oauth_verifier"; //
JDevlin

        protected const string HMACSHA1SignatureType = "HMAC-SHA1";
        protected const string PlainTextSignatureType = "PLAINTEXT";
        protected const string RSASHA1SignatureType = "RSA-SHA1";

        protected Random random = new Random();

        protected string unreservedChars =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-_.~";

        /// <summary>
        /// Helper function to compute a hash value
        /// </summary>
        /// <param name="hashAlgorithm">The hashing algoirhtm used. If
that algorithm needs some initialization, like HMAC and its
derivatives, they should be initialized prior to passing it to this
function</param>
        /// <param name="data">The data to hash</param>
        /// <returns>a Base64 string of the hash value</returns>
        private string ComputeHash(HashAlgorithm hashAlgorithm, string
data)
        {
            if (hashAlgorithm == null)
            {
                throw new ArgumentNullException("hashAlgorithm");
            }

            if (string.IsNullOrEmpty(data))
            {
                throw new ArgumentNullException("data");
            }

            byte[] dataBuffer =
System.Text.Encoding.ASCII.GetBytes(data);
            byte[] hashBytes = hashAlgorithm.ComputeHash(dataBuffer);

            return Convert.ToBase64String(hashBytes);
        }

        /// <summary>
        /// Internal function to cut out all non oauth query string
parameters (all parameters not begining with "oauth_")
```

```
        /// </summary>
        /// <param name="parameters">The query string part of the
Url</param>
        /// <returns>A list of QueryParameter each containing the
parameter name and value</returns>
        private List<QueryParameter> GetQueryParameters(string
parameters)
        {
            if (parameters.StartsWith("?"))
            {
                parameters = parameters.Remove(0, 1);
            }

            List<QueryParameter> result = new List<QueryParameter>();

            if (!string.IsNullOrEmpty(parameters))
            {
                string[] p = parameters.Split('&');
                foreach (string s in p)
                {
                    // jmd: don't strip out oauth_verifier
                    if (!string.IsNullOrEmpty(s) &&
!s.StartsWith(OAuthParameterPrefix))
                    {
                        if (s.IndexOf('=') > -1)
                        {
                            string[] temp = s.Split('=');
                            result.Add(new QueryParameter(temp[0],
temp[1]));
                        }
                        else
                        {
                            result.Add(new QueryParameter(s,
string.Empty));
                        }
                    }
                }
            }

            return result;
        }

        /// <summary>
        /// This is a different Url Encode implementation since the
default .NET one outputs the percent encoding in lower case.
        /// While this is not a problem with the percent encoding spec,
it is used in upper case throughout OAuth
        /// </summary>
        /// <param name="value">The value to Url encode</param>
        /// <returns>Returns a Url encoded string</returns>
        protected string UrlEncode(string value)
        {
            StringBuilder result = new StringBuilder();
            foreach (char symbol in value)
            {
                if (unreservedChars.IndexOf(symbol) != -1)
                {
                    result.Append(symbol);
                }
                else
                {
                    result.Append('%' + String.Format("{0:X2}",
(int)symbol));
                }
            }

            return result.ToString();
        }

        /// <summary>
        /// Normalizes the request parameters according to the spec
        /// </summary>
        /// <param name="parameters">The list of parameters already
sorted</param>
        /// <returns>a string representing the normalized
parameters</returns>
        protected string
NormalizeRequestParameters(IList<QueryParameter> parameters)
        {
            StringBuilder sb = new StringBuilder();
            QueryParameter p = null;
            for (int i = 0; i < parameters.Count; i++)
            {
                p = parameters[i];
                sb.AppendFormat("{0}={1}", p.Name, p.Value);

                if (i < parameters.Count - 1)
                {
                    sb.Append("&");
                }
            }

            return sb.ToString();
        }

        /// <summary>
        /// Generate the signature base that is used to produce the
signature
        /// </summary>
        /// <param name="url">The full url that needs to be signed
including its non OAuth url parameters</param>
        /// <param name="consumerKey">The consumer key</param>
        /// <param name="token">The token, if available. If not
available pass null or an empty string</param>
        /// <param name="tokenSecret">The token secret, if available.
If not available pass null or an empty string</param>
```

```
        /// <param name="httpMethod">The http method used. Must be a
valid HTTP method verb (POST,GET,PUT, etc)</param>
        /// <param name="signatureType">The signature type. To use the
default values use <see
cref="OAuthBase.SignatureTypes">OAuthBase.SignatureTypes</see>.</param>
        /// <returns>The signature base</returns>
        public string GenerateSignatureBase(Uri url, string
consumerKey, string token, string tokenSecret, string httpMethod,
string timeStamp, string nonce, string PIN, string signatureType, out
string normalizedUrl, out string normalizedRequestParameters)
        {
            if (token == null)
            {
                token = string.Empty;
            }

            if (tokenSecret == null)
            {
                tokenSecret = string.Empty;
            }

            if (string.IsNullOrEmpty(consumerKey))
            {
                throw new ArgumentNullException("consumerKey");
            }

            if (string.IsNullOrEmpty(httpMethod))
            {
                throw new ArgumentNullException("httpMethod");
            }

            if (string.IsNullOrEmpty(signatureType))
            {
                throw new ArgumentNullException("signatureType");
            }

            normalizedUrl = null;
            normalizedRequestParameters = null;

            List<QueryParameter> parameters =
GetQueryParameters(url.Query);
            parameters.Add(new QueryParameter(OAuthVersionKey,
OAuthVersion));
            parameters.Add(new QueryParameter(OAuthNonceKey, nonce));
            parameters.Add(new QueryParameter(OAuthTimestampKey,
timeStamp));
            parameters.Add(new QueryParameter(OAuthSignatureMethodKey,
signatureType));
            parameters.Add(new QueryParameter(OAuthConsumerKeyKey,
consumerKey));

            if (!string.IsNullOrEmpty(token))
            {
                parameters.Add(new QueryParameter(OAuthTokenKey,
token));
            }

            // JDevlin: support for PIN-based auth
            if (!String.IsNullOrEmpty(PIN))
            {
                parameters.Add(new QueryParameter(OAuthVerifierKey,
PIN));
            }

            parameters.Sort(new QueryParameterComparer());

            normalizedUrl = string.Format("{0}://{1}", url.Scheme,
url.Host);
            if (!((url.Scheme == "http" && url.Port == 80) ||
(url.Scheme == "https" && url.Port == 443)))
            {
                normalizedUrl += ":" + url.Port;
            }

            normalizedUrl += url.AbsolutePath;
            normalizedRequestParameters =
NormalizeRequestParameters(parameters);

            StringBuilder signatureBase = new StringBuilder();
            signatureBase.AppendFormat("{0}&", httpMethod.ToUpper());
            signatureBase.AppendFormat("{0}&",
UrlEncode(normalizedUrl));
            signatureBase.AppendFormat("{0}",
UrlEncode(normalizedRequestParameters));

            return signatureBase.ToString();
        }

        /// <summary>
        /// Generate the signature value based on the given signature
base and hash algorithm
        /// </summary>
        /// <param name="signatureBase">The signature based as produced
by the GenerateSignatureBase method or by any other means</param>
        /// <param name="hash">The hash algorithm used to perform the
hashing. If the hashing algorithm requires initialization or a key it
should be set prior to calling this method</param>
        /// <returns>A base64 string of the hash value</returns>
        public string GenerateSignatureUsingHash(string signatureBase,
HashAlgorithm hash)
        {
            return ComputeHash(hash, signatureBase);
        }

        /// <summary>
        /// Generates a signature using the HMAC-SHA1 algorithm
        /// </summary>
        /// <param name="url">The full url that needs to be signed
including its non OAuth url parameters</param>
        /// <param name="consumerKey">The consumer key</param>
        /// <param name="consumerSecret">The consumer seceret</param>
        /// <param name="token">The token, if available. If not
available pass null or an empty string</param>
        /// <param name="tokenSecret">The token secret, if available.
If not available pass null or an empty string</param>
        /// <param name="httpMethod">The http method used. Must be a
valid HTTP method verb (POST,GET,PUT, etc)</param>
        /// <returns>A base64 string of the hash value</returns>
        public string GenerateSignature(Uri url, string consumerKey,
string consumerSecret, string token, string tokenSecret, string
httpMethod, string timeStamp, string nonce, /* JDevlin*/ string PIN,
out string normalizedUrl, out string normalizedRequestParameters)
        {
            return GenerateSignature(url, consumerKey, consumerSecret,
token, tokenSecret, httpMethod, timeStamp, nonce, PIN,
SignatureTypes.HMACSHA1, out normalizedUrl, out
normalizedRequestParameters);
        }

        /// <summary>
        /// Generates a signature using the specified signatureType
        /// </summary>
        /// <param name="url">The full url that needs to be signed
including its non OAuth url parameters</param>
        /// <param name="consumerKey">The consumer key</param>
        /// <param name="consumerSecret">The consumer seceret</param>
        /// <param name="token">The token, if available. If not
available pass null or an empty string</param>
        /// <param name="tokenSecret">The token secret, if available.
If not available pass null or an empty string</param>
        /// <param name="httpMethod">The http method used. Must be a
valid HTTP method verb (POST,GET,PUT, etc)</param>
        /// <param name="signatureType">The type of signature to
use</param>
        /// <returns>A base64 string of the hash value</returns>
        public string GenerateSignature(Uri url, string consumerKey,
string consumerSecret, string token, string tokenSecret, string
httpMethod, string timeStamp, string nonce, string PIN /*JDevlin*/,
SignatureTypes signatureType, out string normalizedUrl, out string
normalizedRequestParameters)
        {
            normalizedUrl = null;
            normalizedRequestParameters = null;

            switch (signatureType)
            {
                case SignatureTypes.PLAINTEXT:
                    return
HttpUtility.UrlEncode(string.Format("{0}&{1}", consumerSecret,
tokenSecret));
                case SignatureTypes.HMACSHA1:
                    string signatureBase = GenerateSignatureBase(url,
consumerKey, token, tokenSecret, httpMethod, timeStamp, nonce, PIN,
HMACSHA1SignatureType, out normalizedUrl, out
normalizedRequestParameters);

                    HMACSHA1 hmacsha1 = new HMACSHA1();
                    hmacsha1.Key =
Encoding.ASCII.GetBytes(string.Format("{0}&{1}",
UrlEncode(consumerSecret), string.IsNullOrEmpty(tokenSecret) ? "" :
UrlEncode(tokenSecret)));

                    return GenerateSignatureUsingHash(signatureBase,
hmacsha1);
                case SignatureTypes.RSASHA1:
                    throw new NotImplementedException();
                default:
                    throw new ArgumentException("Unknown signature
type", "signatureType");
            }
        }

        /// <summary>
        /// Generate the timestamp for the signature
        /// </summary>
        /// <returns></returns>
        public virtual string GenerateTimeStamp()
        {
            // Default implementation of UNIX time of the current UTC
time
            TimeSpan ts = DateTime.UtcNow - new DateTime(1970, 1, 1, 0,
0, 0, 0);
            return Convert.ToInt64(ts.TotalSeconds).ToString();
        }

        /// <summary>
        /// Generate a nonce
        /// </summary>
        /// <returns></returns>
        public virtual string GenerateNonce()
        {
            // Just a simple implementation of a random number between
123400 and 9999999
            return random.Next(123400, 9999999).ToString();
        }

    }
}
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Net;
using System.IO;
using System.Collections.Specialized;

namespace CodingTheTweet
{
    public class oAuthTwitter : OAuthBase
    {
        public enum Method { GET, POST };
        public const string REQUEST_TOKEN =
"http://twitter.com/oauth/request_token";
        public const string AUTHORIZE =
"http://twitter.com/oauth/authorize";
        public const string ACCESS_TOKEN =
"http://twitter.com/oauth/access_token";

        private string _consumerKey = "";
        private string _consumerSecret = "";
        private string _token = "";
        private string _tokenSecret = "";
        private string _pin = ""; // JDevlin

        // JMD: this property should not have a dependency on the
Settings file.
        public string ConsumerKey
        {
            get
            {
                //if (_consumerKey == null || _consumerKey.Length == 0)
                //{
                //    _consumerKey = Settings1.Default.consumerKey;
                //}
                return _consumerKey;
            }
            set { _consumerKey = value; }
        }

        // JMD: this property should not have a dependency on the
Settings file.
        public string ConsumerSecret {
            get {
                //if (_consumerSecret.Length == 0)
                //{
                //    _consumerSecret =
Settings1.Default.consumerSecret;
                //}
                return _consumerSecret;
            }
            set { _consumerSecret = value; }
        }

        public string OAuthToken { get; set; }
        public string Token { get { return _token; } set { _token =
value; } }
        public string PIN { get { return _pin; } set { _pin = value; }
}
```

```
        public string TokenSecret { get { return _tokenSecret; } set {
_tokenSecret = value; } }

        /// <summary>
        /// Get the link to Twitter's authorization page for this
application.
        /// </summary>
        /// <returns>The url with a valid request token, or a null
string.</returns>
        public string AuthorizationLinkGet()
        {
            string ret = null;

            // First let's get a REQUEST token.
            string response = oAuthWebRequest(Method.GET,
REQUEST_TOKEN, String.Empty);
            if (response.Length > 0)
            {
                //response contains token and token secret.  We only
need the token.
                NameValueCollection qs =
HttpUtility.ParseQueryString(response);
                if (qs["oauth_token"] != null)
                {
                    OAuthToken = qs["oauth_token"]; // tuck this away
for later
                    ret = AUTHORIZE + "?oauth_token=" +
qs["oauth_token"];// +"&oauth_callback=oob";
                }
            }
            return ret;
        }

        /// <summary>
        /// Exchange the request token for an access token.
        /// </summary>
        /// <param name="authToken">The oauth_token is supplied by
Twitter's authorization page following the callback.</param>
        public void AccessTokenGet(string authToken, string PIN)
        {
            this.Token = authToken;
            this._pin = PIN; // JDevlin

            string response = oAuthWebRequest(Method.GET, ACCESS_TOKEN,
String.Empty);

            if (response.Length > 0)
            {
                //Store the Token and Token Secret
                NameValueCollection qs =
HttpUtility.ParseQueryString(response);
                if (qs["oauth_token"] != null)
                {
                    this.Token = qs["oauth_token"];
                }
                if (qs["oauth_token_secret"] != null)
                {
                    this.TokenSecret = qs["oauth_token_secret"];
                }
            }
        }

        /// <summary>
        /// Submit a web request using oAuth.
        /// </summary>
        /// <param name="method">GET or POST</param>
        /// <param name="url">The full url, including the 
querystring.</param>
        /// <param name="postData">Data to post (querystring 
format)</param>
        /// <returns>The web server response.</returns>
        public string oAuthWebRequest(Method method, string url, string 
postData)
        {
            string outUrl = "";
            string querystring = "";
            string ret = "";


            //Setup postData for signing.
            //Add the postData to the querystring.
            if (method == Method.POST)
            {
                if (postData.Length > 0)
                {
                    //Decode the parameters and re-encode using the 
oAuth UrlEncode method.
                    NameValueCollection qs = 
HttpUtility.ParseQueryString(postData);
                    postData = "";
                    foreach (string key in qs.AllKeys)
                    {
                        if (postData.Length > 0)
                        {
                            postData += "&";
                        }
                        qs[key] = HttpUtility.UrlDecode(qs[key]);
                        qs[key] = this.UrlEncode(qs[key]);
                        postData += key + "=" + qs[key];

                    }
                    if (url.IndexOf("?") > 0)
                    {
                        url += "&";
                    }
                    else
                    {
                        url += "?";
                    }
                    url += postData;
                }
            }
            else if (method == Method.GET &&
!String.IsNullOrEmpty(postData))
            {
                url += "?" + postData;
            }

            Uri uri = new Uri(url);

            string nonce = this.GenerateNonce();
            string timeStamp = this.GenerateTimeStamp();

            //Generate Signature
            string sig = this.GenerateSignature(uri,
                this.ConsumerKey,
                this.ConsumerSecret,
                this.Token,
                this.TokenSecret,
                method.ToString(),
                timeStamp,
                nonce,
                this.PIN,
                out outUrl,
                out querystring);

            querystring += "&oauth_signature=" +
HttpUtility.UrlEncode(sig);

            //Convert the querystring to postData
            if (method == Method.POST)
            {
                postData = querystring;
                querystring = "";
            }

            if (querystring.Length > 0)
            {
                outUrl += "?";
            }

            ret = WebRequest(method, outUrl +  querystring, postData);

            return ret;
        }

        /// <summary>
        /// Web Request Wrapper
        /// </summary>
        /// <param name="method">Http Method</param>
        /// <param name="url">Full url to the web resource</param>
        /// <param name="postData">Data to post in querystring
format</param>
        /// <returns>The web server response.</returns>
        public string WebRequest(Method method, string url, string
postData)
        {
            HttpWebRequest webRequest = null;
            StreamWriter requestWriter = null;
            string responseData = "";

            webRequest = System.Net.WebRequest.Create(url) as
HttpWebRequest;
            webRequest.Method = method.ToString();
            webRequest.ServicePoint.Expect100Continue = false;
            //webRequest.UserAgent  = "Identify your application
please.";
            //webRequest.Timeout = 20000;

            if (method == Method.POST)
            {
                webRequest.ContentType = "application/x-www-form-
urlencoded";

                //POST the data.
                requestWriter = new
StreamWriter(webRequest.GetRequestStream());
                try
                {
                    requestWriter.Write(postData);
                }
                catch
                {
                    throw;
                }
                finally
                {
                    requestWriter.Close();
                    requestWriter = null;
                }
            }

            responseData = WebResponseGet(webRequest);

            webRequest = null;

            return responseData;

        }

        /// <summary>
        /// Process the web response.
        /// </summary>
        /// <param name="webRequest">The request object.</param>
        /// <returns>The response data.</returns>
        public string WebResponseGet(HttpWebRequest webRequest)
        {
            StreamReader responseReader = null;
            string responseData = "";

            try
            {
                responseReader = new
StreamReader(webRequest.GetResponse().GetResponseStream());
                responseData = responseReader.ReadToEnd();
            }
            catch
            {
                throw;
            }
            finally
            {
                webRequest.GetResponse().GetResponseStream().Close();
                responseReader.Close();
                responseReader = null;
            }

            return responseData;
        }

        // JMD: added for convenience. Reset the state of the
oAuthTwitter object.
        public void Reset()
        {
            ConsumerKey = ConsumerSecret = OAuthToken = Token =
TokenSecret = PIN = String.Empty;
        }
    }
}
```

APPENDIX B. EMAIL BASED APPLICATION

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;
using System.Net.Security;
using System.Net.Mail;
using System.IO;

namespace DedicatedEmailAccount
{
    public partial class Form1 : Form
    {
        Email MyEmail = new Email();

        public Form1()
        {
            InitializeComponent();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            try
            {
                //declare variables
                int messagecount = 0;
                string ListMessage, MessageBody, DialogueText = "";
                string MyMessageBody = "";
                string ConversionAmount;
                string From = "";

                string MessageIsFrom = "";
                string OriginalMessage = "";


                // Create a TCP client for a TCP connection
                TcpClient tcpClient = new TcpClient();

                //server and authentication information
                string PopServer = "pop3.live.com";
                string PortNumber = "995";
                string UserName = "convertertest@hotmail.com";
                string PassWord = "2simple";
                StreamWriter sw;
                //create the directories
                if (!Directory.Exists("c:\\UMSEE2"))
                    Directory.CreateDirectory("c:\\UMSEE2");
                if (!File.Exists("c:\\UMSEE2\\usersfile.txt"))
                {
                    sw = new StreamWriter("c:\\UMSEE2\\usersfile.txt");
                    sw.Close();
                }


                //Begin talking to the POP3 Server using the Post
Office Protocol (POP)
                DialogueText = "I say:\r\nConnect me to " + PopServer +
":" + PortNumber + "\r\n\r\n";
                // Connect this TCP client to the server IP/name and
port specified in the form
                tcpClient.Connect(PopServer,
Convert.ToInt32(PortNumber));
                // Create a network stream to retrieve data from the
TCP client
                // Create a secure network stream to retrieve data from
the TCP client
                SslStream netStream = new
SslStream(tcpClient.GetStream());
                netStream.AuthenticateAsClient(PopServer);
                //NetworkStream netStream = tcpClient.GetStream();
                // We need a stream reader to be able to read the
network stream
                System.IO.StreamReader strReader = new
System.IO.StreamReader(netStream);
                // If the connection was made successfully
                int Hour, Month = 0;
                Hour = DateTime.Now.Hour;
                Month = DateTime.Now.Month;
                // if(Month <= 9)


                if (tcpClient.Connected)
                {

                   // MessageBox.Show("Connected, now attempting to
authenticate...");
                    //      ,
DateTime.Now.TimeOfDay.ToString().Substring(0, 5).Replace(":",""));
//);
                    timer1.Enabled = false;
                    DialogueText += "Server says:\r\n" +
strReader.ReadLine() + "\r\n\r\n";
                    // Buffer to which we're going to write the
commands
                    byte[] WriteBuffer = new byte[1024];
                    // We're passing ASCII characters
                    ASCIIEncoding enc = new
System.Text.ASCIIEncoding();
                    // Pass the username to the server
                    WriteBuffer = enc.GetBytes("USER " + UserName +
"\r\n");
```

```
                    DialogueText += "I say:\r\nHere's the username: " +
UserName + "\r\n\r\n";
                    netStream.Write(WriteBuffer, 0,
WriteBuffer.Length);
                    DialogueText += "Server says\r\n" +
strReader.ReadLine() + "\r\n\r\n";
                    // Pass the password to the server
                    WriteBuffer = enc.GetBytes("PASS " + PassWord +
"\r\n");
                    DialogueText += "I say:\r\nHere's the password: " +
PassWord + "\r\n\r\n";
                    netStream.Write(WriteBuffer, 0,
WriteBuffer.Length);
                    DialogueText += "Server says:\r\n" +
strReader.ReadLine() + "\r\n\r\n";
                    // Now that we are (probably) authenticated, list
the messages
                    WriteBuffer = enc.GetBytes("LIST\r\n");
                    DialogueText += "I say:\r\nPlease list the
messages\r\n\r\n";
                    netStream.Write(WriteBuffer, 0,
WriteBuffer.Length);
                    while (true)
                    {
                        ListMessage = strReader.ReadLine();

                        if (ListMessage == ".") // if this is the last
(most recent) message on the server.
                        {

                            // It's the last message so display all
info for it and exit the loop
                            WriteBuffer = enc.GetBytes("RETR " +
Convert.ToString(messagecount - 1) + "\r\n");
                            netStream.Write(WriteBuffer, 0,
WriteBuffer.Length);
                            MessageBody = "";
                            while (MessageBody != ".") //
                            {
                                // Get all information  for the most
recent information, line-by-line
                                MessageBody = strReader.ReadLine();
                                DialogueText += "Server says\r\n" +
MessageBody + "\r\n\r\n";
                                //  messageLineCount++;
                                // messageLineArray[messageLineCount] =
MessageBody;
                                MyMessageBody += MessageBody;

                            }

                            break;
                        }
                        else
                        {
                            // List the message summary (not the entire
message.)
                            messagecount++;// keep a count of how many
messages there are.
                            DialogueText += "Server says:\r\n" +
ListMessage + "\r\n\r\n";
                            continue;
                        }
                    }

                    DialogueText += "I say:\r\nThanks, we will
disconnect now\r\n\r\n";
                    WriteBuffer = enc.GetBytes("DELE " +
Convert.ToString(messagecount - 1) + "\r\n");
                    netStream.Write(WriteBuffer, 0,
WriteBuffer.Length);


                    DialogueText += "I say:\r\nThanks, we will
disconnect now\r\n\r\n";
                    WriteBuffer = enc.GetBytes("QUIT\r\n");
                    netStream.Write(WriteBuffer, 0,
WriteBuffer.Length);
                    DialogueText += "Server says:\r\n" +
strReader.ReadLine();

                    OriginalMessage = MyMessageBody;

                    MessageIsFrom =
MyMessageBody.Substring(MyMessageBody.IndexOf("From:") + 5);
                    string LocalPart = MessageIsFrom.Substring(0,
MessageIsFrom.IndexOf('@')).Trim();
                    string DomainName =
MessageIsFrom.Substring(MessageIsFrom.IndexOf(@"@"));
                    DomainName = DomainName.Substring(0,
DomainName.IndexOf("To:"));
                    MessageIsFrom = (LocalPart + DomainName).Trim();

                    // MyMessageBody =
MyMessageBody.Substring(MyMessageBody.IndexOf("quoted-printable") +
16);
                    // MyMessageBody = MyMessageBody.Substring(0,
MyMessageBody.IndexOf("="));


                    //Make everything lower case

                    using (StreamWriter sw1 = new
StreamWriter("c:\\UMSEE2\\EmailString.txt", true))
                    {
                        sw1.WriteLine("Local Part: " + LocalPart);
                        sw1.WriteLine("DomainName: " + DomainName);
                        sw1.WriteLine(MyMessageBody);
```

```
                                sw1.WriteLine("\n\n");
                                sw1.Close();
                            }//end using
                            /* if (MyMessageBody.Contains("reg"))
                                 reg(MessageIsFrom);
                             else if (MyMessageBody.Contains("mfile"))
                                 CreateMfile(MessageIsFrom);*/
                            MyMessageBody = MyMessageBody.ToLower();
                           // MessageBox.Show("Checking Message");
                            if (OriginalMessage.Contains("Content-Type: image")
&& MyMessageBody.Contains("jpg"))
                                MyEmail.ExtractImage(MessageIsFrom,
OriginalMessage);
                            else if (MyMessageBody.Contains("reg1 "))
                            {
                               // MessageBox.Show("Contains Reg");
                                MyEmail.RegUser(MessageIsFrom, MyMessageBody);
                            }
                            else if (MyMessageBody.Contains("mfile") &&
MyMessageBody.Contains("??"))
                                MyEmail.CreateMFile(MessageIsFrom,
OriginalMessage);
                            else if (MyMessageBody.Contains("parab"))
                                MyEmail.Parab(MessageIsFrom, OriginalMessage);
                            else if (MyMessageBody.Contains("view entire
message"))
                                MyEmail.GetSprintImage(MessageIsFrom,
OriginalMessage);
                            else if (MyMessageBody.Contains("hlpreg"))
                                    MyEmail.HlpReg(MessageIsFrom);
                            else if (MyMessageBody.Contains("hlpgraph"))
                                    MyEmail.HlpGraph(MessageIsFrom);
                            else if (MyMessageBody.Contains("hlpmfileval"))
                                    MyEmail.HlpMfile(MessageIsFrom);



                            timer1.Enabled = true;
                            /* sw = new
StreamWriter("c:\\UMSEE2\\usersfile.txt");
                              sw.WriteLine("From: " + MessageIsFrom);
                              sw.WriteLine(DateTime.Now);`
                             sw.Write(MyMessageBody);
                             sw.Close();*/

                            /*  if (MyMessageBody.Contains("Content-
Disposition: attachment;"))
                              {
                                  //extracts the name of attachment


                              }//end if*/
                        }//end if
```

```
                else //Otherwise, if the tcip connection was not made
successfully then do the following...
                {
                    Console.WriteLine("Connection could not be made!
Please try entering information again.\n");
                }

            }//end try
            catch (Exception ex)
            {
               // MessageBox.Show("There was a catch");
                MessageBox.Show(ex.ToString());
                timer1.Enabled = true;
            }//end catch
        }//end timer1_Tick
    }
}
```

APPENDIX C. OCTAVE TOOLBOX

```
function runmfile ()

cd c:\UMSEE\MFILE
figure;
##fig = gcf()
## Grad the latest m file
d = dir(["c:\\UMSEE\\MFILE\\*"]);
try
	if(length(d) ~= 0)
		file_name = d(1).name;
		##Save Text From MFile
		file = fopen(file_name, 'r');
		A = fscanf(file, '%s');
		fclose(file);
		name = substr(A, 3, index(A, ';') - 3)
		number = substr(A,index(A, ';') + 1);
		number = substr(number,3, index(number, ';') - 3)
		mytry = substr(file_name, 1, index(file_name, ".m") - 1)
		eval(mytry,'printf ("This error occurred:\n%s\n", lasterr ());' );
		##if (file_name =="threed.m")

		##else
		delete(file_name);
		#3endif
		gcf
		##Send the figure

		cd c:\UMSEE
		print -djpg ImageToSend.jpg
		saveToFile = [';;;' 'c:\\UMSEE\\ImageToSend.jpg;' name ';' number
';image;' lasterr()];
		##Save the info to file
		fid = fopen('InterfacePicture.txt', 'w');
		fprintf(fid,saveToFile );
		fclose(fid);
		system(['MatlabToolBoxEmailPic.application']);
	endif

catch
	cd c:\UMSEE

end
endfunction
```

```
#function SEND_PIC ()
#cd c:\UMSEE
##Call The C# Function-Email Off The Data
#system(['MatlabToolBoxEmailPic.application']);
#endfunction

function send_pic (I,username)

##try to send from ncat account
##call the program
cd c:\UMSEE
try
        imwrite(I,'anyImage.jpg');
catch
end
file = fopen('usersfile.txt', 'r');
A = fscanf(file, '%s');
fclose(file);

file2 = fopen('lasterr.txt', 'r');
lasterr = fscanf(file2, '%s');
fclose(file2);

number = substr(A,index(A, username) + length(username));
number = substr(number,1,index(number,';') - 1)
##write to file
saveToFile = [';;;' 'c:\\UMSEE\\anyImage.jpg;' username ';' number ';image;' lasterr];
##Save the info to file
fid2 = fopen('InterfacePicture.txt', 'w');
fprintf(fid2,saveToFile );
fclose(fid2);
NCAT_Account = 1
##Call The C# Function-Email Off The Data
system(['MatlabToolBoxEmailPic.application']);
endfunction


function  [ i, username] = get_pic ();
try
        ##Change To Text Directory
        cd Image
        ##name = input('Enter the account name ' ,"s");
        d = dir(["c:\\UMSEE\\Image\\*"]);
```

```
##Load The Last Sent Image
file_name = d(1).name
i = imread(d(1).name);
imwrite(i, 'image.jpg');
mytry = substr(file_name, 1, index(file_name, ".") - 1);
eval(mytry,'printf ("This error occurred:\n%s\n", lasterr ());' );
##Delete The Image/Switch To Normal Directory
##if d(1).name ~= 'image.jpg'
delete(d(1).name);
##endif
cd c:\UMSEE;

##Extract and Alter The Image and Info


##Determine User Info
cd c:\UMSEE\Image
Command_File = d(1).name;
Command_File = substr(Command_File, 1, index(Command_File, '.') - 1);
Command_File = [Command_File '.m'];
file = fopen(Command_File, 'r');
A = fscanf(file, '%s');
fclose(file);
##if Command_File ~= 'threed.m'
delete(Command_File);
##endif
cd c:\UMSEE
name = substr(A, 3, index(A, ';') - 3);
number = substr(A,index(A, ';') + 1);
number = substr(number,3, index(number, ';') - 3);
username = name;

##start_index = index(A, name) + length(name);
##number = substr(A, start_index);
##number = substr(number, 1, index(number, ';') - 1);
##saveToFile = [';;;' 'c:\\UMSEE\\image.jpg;' name ';' number ';image;' lasterr];

##Save the Processed image
##print -djpg image.jpg
print -djpg anyImage.jpg

##Save the info to file
fid = fopen('lasterr.txt', 'w');
```

```
        fprintf(fid,lasterr );
        fclose(fid);
catch
        cd c:\UMSEE
end



endfunction
```