



Project Number: DRB MQP 1100



“TweetSign”

A Scrolling LED Twitter Display

A Major Qualifying Project Report submitted
to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for
the Degree of Bachelor of Science by

Andrew Creeth

Ajay Dhesikan

Date: March 2, 2012

Professor Donald R. Brown

Project Advisor & Mentor

Abstract

The purpose of this project was to develop a scrolling LED sign with an embedded system capable of wirelessly interfacing with the Twitter social networking service. The embedded system is built on an ASUS wireless router running a specialized version of Linux. Python code running on the router downloads information through the Twitter API and updates the sign over an RS-232 serial interface. The sign is configured over local network via a web browser by accessing the system's PHP-enabled web server. Authenticated access is achieved by connecting to a third-party server that requests access from Twitter. The final product supports a variety of configurable operating modes.

Acknowledgements

Our project would not have been possible without the help of our advisor, Professor Donald R. Brown. He provided us with guidance while still allowing us to make our own design and implementation decisions. We thank Professor Brown for all of his comments, suggestions, and assistance.

One of the biggest missing pieces of our team was in mechanical design. While we had visions of how the overall product should look, we did not have the means or knowledge to make the visions a reality. Our friend Ennio Claretti provided assistance in this area. He spent many hours working with us on the SolidWorks model and using the laser cutter to help us build a more professional final product than we would have been able to achieve on our own. We owe him our thanks for all of his help on this project.

We would also like to thank the professors and students who worked on the MIT Center for Civic Media's Realtime Community Signage project¹. Their source code served as the basis for developing our drivers for the interface between the embedded system and the LED sign.

¹ <http://civic.mit.edu/blog/rahulb/realtime-community-signage-source-code>

Table of Contents

Abstract.....	i
Acknowledgements.....	ii
1 Introduction.....	1
2 Background.....	3
2.1 Twitter.....	3
2.2 Important Twitter Terminology.....	3
2.3 Twitter Being Used For Marketing.....	6
2.4 Competition.....	9
2.5 Requirements and Features.....	11
2.6 Basic Hardware Layout.....	13
2.7 Choosing a Scrolling LED Sign.....	14
2.8 Contents and Details of the Chosen Scrolling LED Sign.....	15
2.9 Choosing an Embedded Platform.....	19
2.9.1 Mini-ITX.....	19
2.9.2 Arduino.....	20
2.9.3 Off-The-Shelf Router.....	22
2.9.4 Value Analysis.....	23
2.10 Choosing a Router Model.....	26
2.11 The Three Available Twitter APIs (REST, Search, Streaming).....	28
2.12 Choosing a Programming Language.....	34
3 Final Design.....	36
3.1 High Level Design Overview.....	36
3.2 Features (Operating Modes).....	37
3.3 User Experience.....	38

4	Methodology	39
4.1	Installing OpenWrt	39
4.2	Interfacing with Twitter	41
4.3	Interfacing with the Scrolling Led Sign.....	46
4.4	Physical Integration	46
4.5	Bill of Materials	51
4.6	Testing and Debugging.....	51
5	Results.....	53
6	Conclusion	55
7	Future Work.....	56
8	Appendices	59
8.1	Appendix A – Extended Twitter History and Statistics.....	59
8.2	Appendix B – Scrolling LED Sign Table	62
8.3	Appendix C – Python Script communicating with Twitter through REST API	63
8.4	Appendix D – Comprehensive Guide to OpenWrt.....	64
8.4.1	OpenWrt Image Builder	64
8.4.2	Flashing the Image to the Router	66
8.4.3	First Run Command Line Access	68
8.4.4	Network Configuration	69
8.4.5	Mounting the USB Drive	72
8.4.6	Installing Packages to the USB Drive	72
8.4.7	Web Server and PHP Configuration.....	74
8.4.8	Initialization Scripts.....	74
8.4.9	Further Reading	77
8.5	Appendix E – Getting Started Guide	78

Table of Figures

Figure 1: Simplified comparison of the Twisplay and TweetSign.	2
Figure 2: Various key Twitter terms demonstrated using Jack Dorsey's profile.	6
Figure 3: Software Kitchi's UI screen.	10
Figure 4: Joshua Persky, founder of Twisplays, standing next to his product.....	11
Figure 5: The general hardware layout of the TweetSign.	13
Figure 6: The Sign, Msg, Scroll, 24Chr, Bk LED sign from Amazon for \$404.49.....	14
Figure 7: BETAbrite prism indoor color LED display sign - custom quote from Salescaster set the price at \$250	14
Figure 8: The Ultra-Glow-2™ Indoor Brightness Red 26" LED Sign within the package.	15
Figure 9: The additional contents and the scrolling LED sign.....	16
Figure 10: Holding the LED sign in front of a door in order to show relative size.....	17
Figure 11: Default LED message once the power source is connected.....	18
Figure 12: Custom static message programmed on the LED sign using the provided software..	18
Figure 13: Standard passively cooled Mini-ITX motherboard.	19
Figure 14: Arduino Uno main board. Headers on the edge are for stacking I/O “shields”	21
Figure 15: ASUS WL520GU router.	28
Figure 16: Creating and registering a new application with Twitter. The screenshot does not show the terms of service or captcha image included on the original page.....	32
Figure 17: The details tab of the registered application which shows important information necessary to make requests using OAuth.	32
Figure 18: Authorization screen for applications.	33
Figure 19: Hardware layout for the TweetSign.....	36
Figure 20: Authorization screen for TweetSign.	43
Figure 21: Diagram of how Oauth works on the TweetSign. The starting point is with the setup page (shown in an oval in the left center of the diagram).	45
Figure 22: LED Sign Power Rails and Serial Connection.....	47
Figure 23: LED Sign Serial Interface	48
Figure 24: Cables and Adapters Removed From the TweetSign	49

Figure 25: Preliminary CAD Drawing of the Case.....	50
Figure 26: Laser Cutting the Case	50
Figure 27: The Completed TweetSign.....	53
Figure 28: Back View of the TweetSign.....	53
Figure 29: A screenshot of Twttr's homepage in its early days.....	60
Figure 30: Twitter's adoption rate since its launch in 2006	60
Figure 31: A timeline providing information about select record-breaking tweet events	61
Figure 32: Network configuration in Mac OS X	67
Figure 33: Example wireless configuration file.....	70
Figure 34: LAN section from example network configuration file.....	71
Figure 35: Basic framework of an initialization script	75
Figure 36: Example initialization script from the TweetSign	76
Figure 37: TweetSign USB contents.....	78
Figure 38: TweetSign Network Configuration Utility.....	79
Figure 39: Successful Network Configuration.....	79
Figure 40: Network Configuration Page URL	80
Figure 41: Configuration Homepage.....	80
Figure 42: Basic Setup Page	81
Figure 43: Advanced Settings Page.....	81
Figure 44: Twitter authorization page for TweetSign.....	82
Figure 45: TweetSign Oauth initial success page.....	83

1 Introduction

Twitter grows in popularity every single day. As more and more influential people sign up and start posting updates (tweets), they bring an entirely new set of users into the Twitter ecosystem. With roughly 200 million tweets being sent out each day², it's no doubt that there are a lot of users staying updated with friends, companies, and celebrities. Twitter is one of the best ways to instantly spread information to a large audience. However, one of the biggest barriers that limit Twitter's audience is the fact that it exists solely online in the digital world. This makes it easy to reach people who carry devices with internet access, but less connected segments of the population will be left in the dark. By using signs to bring content from Twitter into the physical world, an entirely new demographic can benefit from the flow of information.

There are a number of situations where businesses and individuals could benefit from using signs to display Twitter content. For example, a sign outside someone's office could be used to update visitors when the individual will be back from lunch, or if they are running late for a meeting. Internet businesses have been successful using Twitter as a platform to advertise new products and promotions. A Twitter sign could be used to give traditional offline businesses the same opportunities. Twitter can be used as a simple and reliable way to update the sign from anywhere on the internet.

Using signs to display Tweets is an idea that has been explored by a handful of people. There are many non-commercial projects online that describe how to build a sign that displays Tweets, but they all require the user to be familiar with electrical engineering. Currently there is only one company attempting to profit from selling Twitter signs: an Internet company called Twisplays that sells scrolling LED signs for \$249. However, the Twisplay has the major disadvantage of requiring an attached computer to receive information from Twitter. This seriously limits the applications of the sign, as running a computer can be expensive and require a lot of power. As a result, the Twisplay can be an impractical or inconvenient option for end

² <http://techcrunch.com/2011/06/30/twitter-3200-million-tweets/>

users. The market is currently lacking an inexpensive sign targeted at non-technical end users that doesn't require anything but a wireless internet connection.

The TweetSign is designed to overcome the limitations of the Twisplay and fill this gap in the market. Unlike the Twisplay, the TweetSign does not require an external computer to function after initial configuration. The TweetSign can use a wireless internet connection to directly interface with Twitter and download the latest Tweets. It is designed to be easily programmed through a web browser without requiring any technical ability. Once the TweetSign is programmed, it can be left alone as it automatically downloads Tweets and updates its display according to the customer's preferences. A simplified comparison of the Twisplay and TweetSign can be seen in Figure 1. The TweetSign is the first Twitter sign designed to create a simple, seamless gateway to bring digital tweets into the physical world.

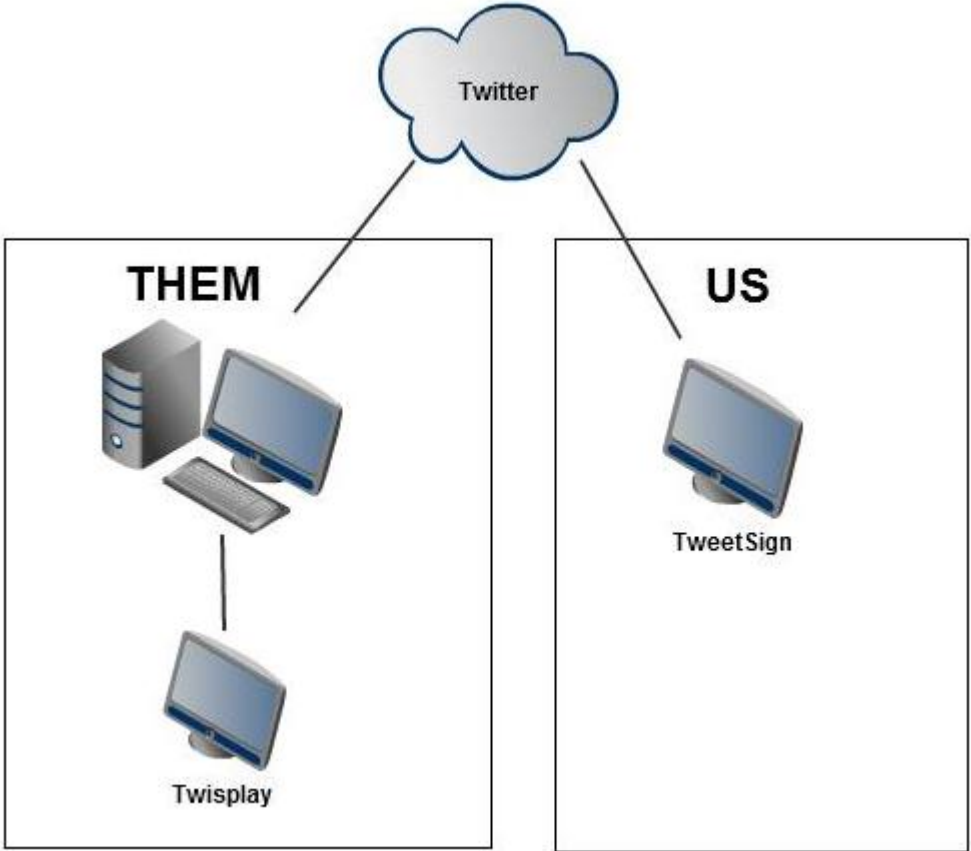


Figure 1: Simplified comparison of the Twisplay and TweetSign.

2 Background

Understanding Twitter, how it is being used, and how to communicate with it via software are all important aspects of this project. This section delves into those topics and also discusses potential competitors, the requirements for the TweetSign, as well as what hardware and software can be used to fulfill these requirements.

2.1 Twitter

Twitter is a unique social networking site built around the answer to one simple question: What are you doing right now? The service allows people to send out 140 character updates known as “Tweets” to their followers worldwide. As of September 2011, Twitter (<http://twitter.com/>) has acquired 200 million registered users.³ It continues to grow significantly, acquiring new users every day. The Twitter audience consists of all sorts of different people, not just those who are technology-savvy. Celebrities, athletes, politicians, and many other influential people attract the general public to join and interact with them. People of varying ages, including many over the age of 50, are using Twitter on a regular basis. Twitter is constantly being mentioned on television and other media outlets.

2.2 Important Twitter Terminology

Twitter has invented many new terms that have become common in the world of Twitter. In order to understand how the various aspects and features of our proposed product work, this terminology is very important to understand. Therefore, the most important terms will be covered briefly in this section.

The Twitter system consists of two groups for each registered user – **Followers and Following**. The following group is often referred to as the friends group. Each user can post a message that is limited to 140 characters. These messages, which are often referred to as posts or updates, are officially called **Tweets**. All of the user’s followers can see these tweets.

³ <http://mashable.com/2011/09/30/twitter-history-infographic/>

Therefore, any updates from any person in a particular user's "following" group will be visible to the user. This is a little confusing at first, but will become clearer as additional terms are introduced.

Every registered user on Twitter must have a username associated with the account. This username is what can be used to find specific users and follow them. People often distinguish Twitter usernames by saying @ ("at") and then the username (combined, it would be something like @username). For example, Jack Dorsey, the executive chairman of Twitter, has the username "jack", which is publicly visible on Twitter as "@jack". People may also refer to @username as a Twitter **Handle**, but officially, the handle is a combination of the username and the Twitter URL (such as <http://twitter.com/jack> for Jack Dorsey).

The **Public Timeline** consists of updates from all users on Twitter who share their tweets publicly. Since there are so many users on Twitter, the public timeline is updated extremely frequently and shows a very large number of tweets every day. Whereas the Public Timeline shows the tweets from all public users in real-time, the **Home Timeline** generally displays only the tweets from those users that a specific person is following (sometimes, however, Twitter also shows tweets from users it thinks may be relevant to those being followed).

Replies (often called "at replies") are responses from one user to another that everyone who has access to the user's stream can see (these tweets always start with the @ sign followed by the username of the person to whom the user is replying). These are generally created by clicking the "Reply" button in the Twitter interface after seeing a tweet that is of interest to a user. For example, if the user Person1 tweeted "How is everyone today?" Another user Person2 can reply to that message publicly by tweeting "@Person1 I'm doing really well! How are you?"

When one user mentions another user anywhere in a tweet by using the @ sign followed by the username, this is known as a **Mention**. Replies, therefore, are inherently Mentions. However, the difference is that Replies must *start* with the @ sign followed by the username whereas Mentions can place that anywhere in the tweet. Another core difference is that Mentions can be made to multiple users in a single post whereas a Reply must be directed to a

single user. An example of using Mentions might be something like, “I had so much fun hanging out with @User1 @User2 and @User3 today!”

Retweets usually start with “RT” (although it *can* be found anywhere in the tweet) and are basically the “forwarding” of a tweet shared by someone else. Retweet can be used as either a noun or a verb. As a noun, it refers to the actual tweet forwarded by another user. As a verb, it refers to the act of forwarding a tweet to other users. For example, if one user Person1 posts something like “The new movie XYZ is out in theaters now!” and another user Person2 finds it interesting and wants to share it with all of his followers, Person2 can retweet this message. The retweeted message would look like “RT @Person1: The new movie XYZ is out in theaters now!” The idea behind it is to spread information to more people by passing on a message. If there are enough characters available, the person who is retweeting can add his own opinion to the tweet. In the previous example, the modified retweet may look like “I want to see it! RT @Person1: The new movie XYZ is out in theaters now!”

Hashtags are “a community-driven convention for adding additional context and metadata to your tweets. You add them in-line to your Twitter posts by prefixing a word with a hash symbol (or number sign).”⁴ (Examples include #TheMostCommonLies, #sandiegofire, #whitecollar, etc.)

By clicking a yellow star next to (below) a tweet, it can be added to a user’s **Favorites** list. These favorites can be later viewed from the “Favorites” tab in the profile section of Twitter. Users may wish to favorite tweets that contain interesting information or URLs.

Twitter has a long list of terms and definitions in their “Twitter Glossary”⁴ which includes many more words and phrases. This section was meant to cover the most relevant ones that will likely reappear in later sections. Select terms have been demonstrated visually in Figure 2.

⁴ <https://support.twitter.com/groups/31-twitter-basics/topics/104-welcome-to-twitter-support/articles/166337-the-twitter-glossary>

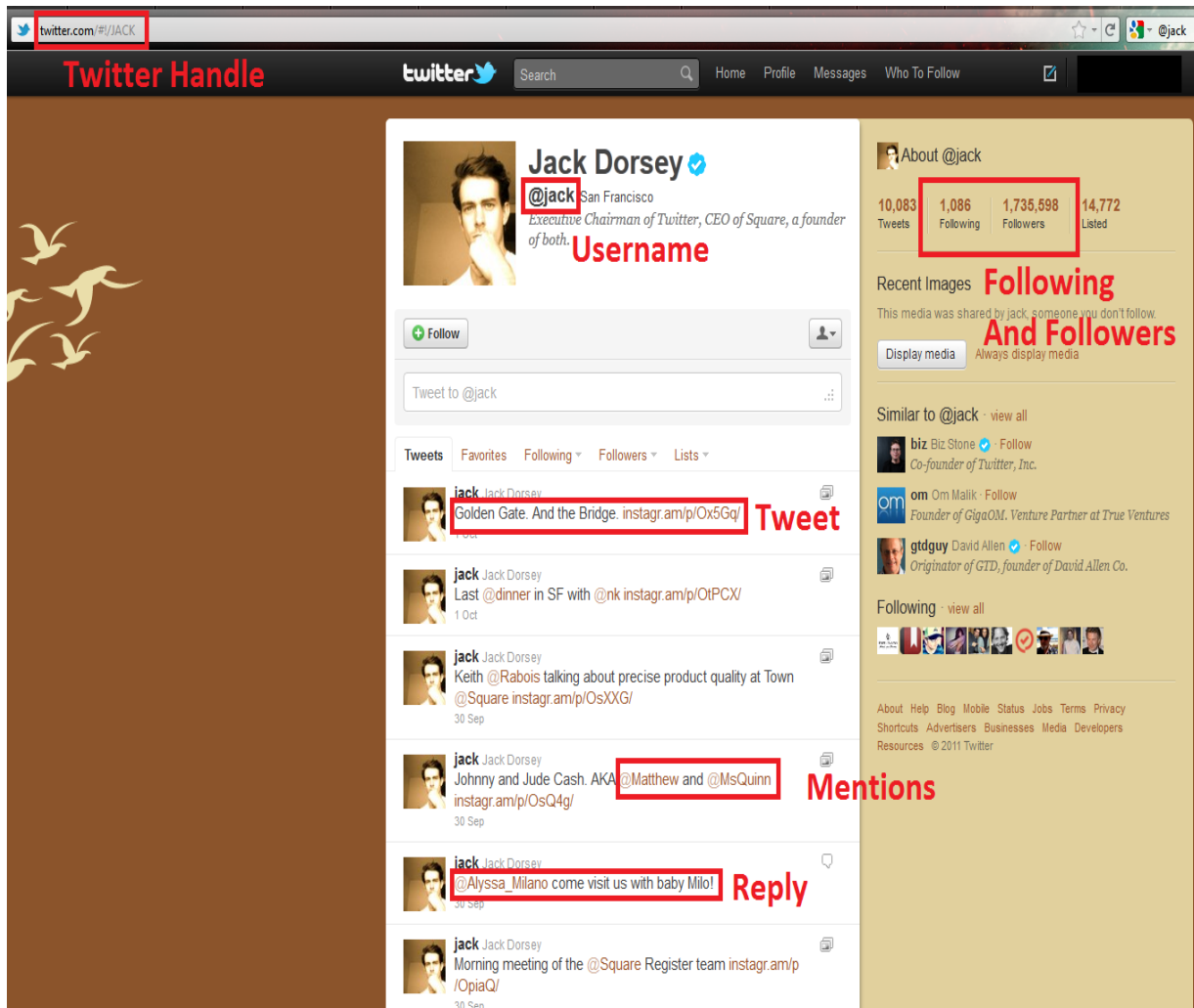


Figure 2: Various key Twitter terms demonstrated using Jack Dorsey's profile.

2.3 Twitter Being Used For Marketing

As Twitter has gained such large traction, businesses, whether large or small, are really starting to pay attention to social media and are leveraging it to promote and grow their own businesses by interacting with the many current users of the site. Some of the relevant uses of

Twitter for businesses have involved finding new jobs, getting referrals, gaining new clients, hiring new employees, building brand loyalty, and selling products.⁵

There have been examples of web developers getting new jobs through answering questions, getting public referrals from friends or past clients, and expanding their networks. Twitter has also been used to find interested clients and stir up conversation with them. Many companies have publicly held important conversations with potential customers to see what they can improve about their products, how they can provide better service, etc. Brand recognition through Twitter is also significant. When customers or clients see constant updates about various products or discounts, they continue coming back to that company.

One interesting approach that some companies have taken is to use Twitter's search features to find people currently talking about products that are related to ones the company has produced, and then tweet them with advice or recommending articles and news information that they may be interested in. For example, "Frank Eliason, Director of Digital Care for Comcast Cable uses Twitter to look for people talking about Comcast, 'tweets' them, and offers to help."⁶

However, Twitter doesn't have to be only for creating new conversations and replying to specific people; it can also be used as a tool for notifying followers about relevant information, such as special events, deals, discounts, contents, statistics, etc. For example, Canal Restaurant (@CanalDining) in Worcester, MA, uses Twitter to provide such information to their followers; one of their tweets was, "Tonight Worcester people 90 Harding Street Canal Nightclub Come tweet your way in and rcv a free frozen bahama mama."⁷ Many of their other tweets are also promotions that customers can take advantage of if they mention Canal Restaurant's Twitter account.

Another example of a company using Twitter to send information out involves the Worcester Consortium (@WorcConsortium), "an alliance of colleges and universities in Central

⁵ <http://gigaom.com/collaboration/real-life-twitter-business-success-stories/>

⁶ http://www.readwriteweb.com/archives/social_media_for_business_who_is_doing_it.php

⁷ <http://twitter.com/#!/canaldining>

Massachusetts.”⁸ They provide brief descriptions about various events, special speakers, contents, and fairs. A few examples⁹ of their informative tweets include:

- “We made the cover of the May issue of Inside Worcester magazine with a story on our student scavenger hunt!” (May 20, 2010)
- “Consortium schools feature some great commencement speakers: Scott Brown (Nichols); Curt Schilling (WPI) and Larry Lucchino (Anna Maria).” (May 12, 2010)
- “The Woo Tube 101 video contest begins Monday, February 1!!! <http://snipurl.com/ta117>” (Jan 27, 2010)

Shrewsbury Health and Racquet Club (@ShrewsburyClub), a place which provides swimming pools, tennis lessons, massages, and various other services, uses Twitter to talk about broad topics and stats related to their primary services. Some of their tweets¹⁰ include:

- "Which contains the most fat? (Answer posted tomorrow) A) One ounce of turkey breast B) One ounce of lean ham C) One ounce of turkey franks" (Oct 13, 2011)
- “Did you know? Next to water, protein makes up the greatest portion of our body weight.” (Oct 9, 2011)

Searching Twitter will reveal many more examples of local businesses using the service to communicate with existing and potential customers.

⁸ <http://www.cowc.org/>

⁹ <http://twitter.com/#!/WorcConsortium>

¹⁰ <http://twitter.com/#!/ShrewsburyClub>

2.4 Products for Public Display of Tweets

In order to obtain a better idea of the potential market and pricing, we set out to explore what projects similar to ours have been previously done. To our surprise, we didn't find many direct competitors. Many of the similar projects we saw were created by individual people and were not available for sale. The one direct competitor we were able to identify is Twisplays¹¹.

As for the local projects, there are many scrolling LED sign examples on YouTube. Many of them are getting data from RSS feeds and displaying that data. The scrolling LED signs used for these projects vary significantly. The main ones seemed to be Eurolite ESN 8x70 Red/Green/Yellow, Alpha PPD220, Alpha 7120C, and BetaBrite. Some projects used two-line signs, while most used one-line signs.

A few examples of these projects can be seen through the following links:

- <http://www.youtube.com/watch?v=BHRuMiLIsBU>
- <http://www.youtube.com/watch?NR=1&v=goQFce5cqLs>
- <http://www.youtube.com/watch?v=BWzECNyTyxA>

One related project, called Kitchi¹², provides software that allows more defined control over certain LED signs. It claims to be able to download RSS feeds (involving news, weather, or other updates) and send them to a scrolling LED sign. The compatibility is with one-line Alpha signs, two-line Alpha signs, and Betabrite signs. In order to use the software, the LED sign must be connected to the computer using either a USB port or a serial port. This software tool is not a direct competitor since we will be providing not only wireless capability, but also hardware and software as part of our product. People may be interested in outputting things to the LED sign that are not Twitter-related.

¹¹ <http://twisplays.com/>

¹² <http://kitchi-rss.com/index.html>

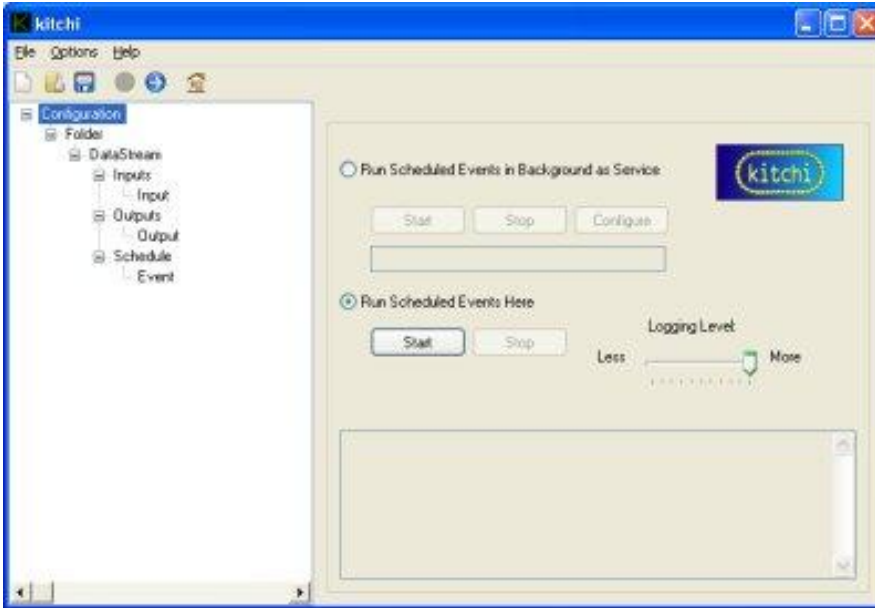


Figure 3: Software Kitchi's UI screen. (Source: http://www.remote-control.net/documentation/kitchi/kitchi_application.jpg)

As previously mentioned, at the time of this writing, the one direct competitor is Twisplays. This product was added to its main website's catalog on September 8, 2010, so it has been around for quite a while. It was originally created by Joshua Persky, who was formerly a NYC banker. He claims to have gotten the idea for Twisplays after seeing all of the LED ticker signs for stocks in New York. Only the wired model of the product is available on the website. It is priced at \$249.00, with \$20.00 for shipping and handling. This was important information for our product, since it gave us a new goal to meet in order to be viable on the market. We set our sights on meeting a retail price of \$249.00 or lower.

Joshua Persky with a single Twisplay unit can be seen in Figure 4. The original Twisplay was said to be 26" by 4", but the dimensions of the product in the picture have not been revealed.¹³ In terms of sales, according to one blog, Joshua has sold "three to a social media restaurant in NYC, and is in negotiations with a telecom company for a massive college campus network. He's hoping to sell 1,000 signs by November 2011."¹⁴

¹³ <http://mashable.com/2010/09/28/twisplays/>

¹⁴ <http://techaccess.com/2011/08/ten-tech-startupsto-keep-an-eye-on/>

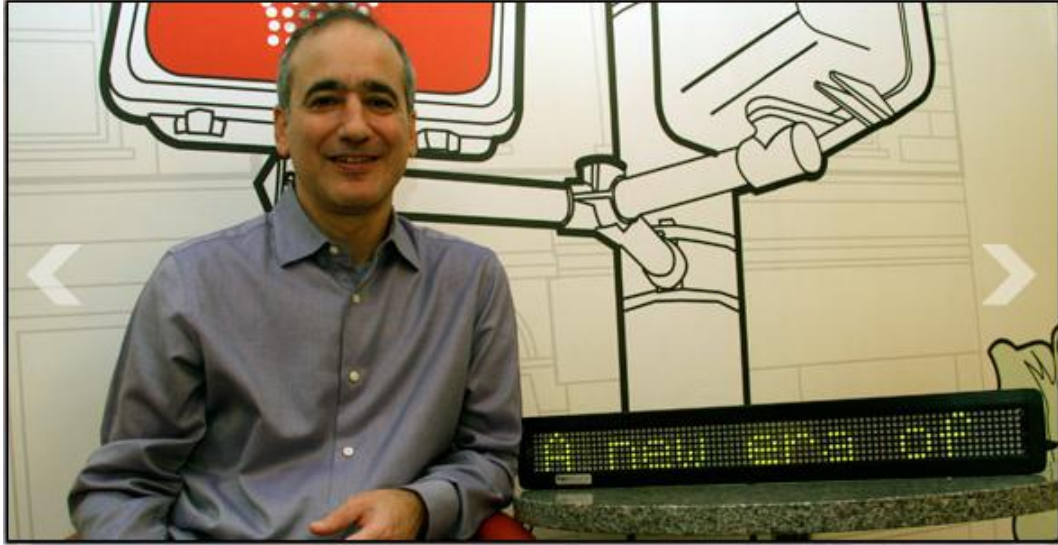


Figure 4: Joshua Persky, founder of Twisplays, standing next to his product. (Source: http://twisplays.com/images/slider/slider_02.jpg)

2.5 TweetSign Requirements and Features

Based on the research we have compiled, there are several core requirements that our product must meet:

1. Wireless capability – This will be a huge distinguisher for our product. The product must be able to connect to unencrypted Wi-Fi networks and WPA/WPA2 encrypted networks.
2. Competitive cost – The product should not be too expensive. Since the TweetSign is wireless, it would justify a higher price than a non-wireless sign like the Twisplay. However, to maintain a competitive edge we are aiming to price the TweetSign at \$249, the same price as the Twisplay.
3. Durable – The product must not break easily.
4. Light – The total weight of the product should not exceed 15 pounds. This product should be able to hang from the wall or rest on a larger window sill.
5. Visible from 20 feet indoors – People should be able to read the sign without having to go directly in front of it.

As far the features of the product go, we have split it into three sections: core functionalities, nice-to-have features, and unsupported features. The core functionalities are essentially software requirements that must be met. Even a barebones product must perform these core functionalities. The nice-to-have features are ones that will selectively get added to the product depending on the amount of time available in the development cycle. The unsupported features include mostly functions that wouldn't make sense to be able to do from a scrolling LED sign. A closer look at these features is provided below.

- **Core Functionalities**

- Unauthenticated user (No complex configuration required, uses the REST API¹⁵)
 - Display random, recent tweets from public users (default configuration)
 - Display latest tweet from one selected public user
- Authenticated user (Using the REST API)
 - Show the latest tweet that would show up in the user's timeline

- **Nice-to-have features**

- Unauthenticated user
 - Display tweets of selected users (scroll through them), not real-time but on some polling interval
- Authenticated user (Using the REST API)
 - Show last X tweets from the user that was authenticated (scroll through them with some interval) – Advantage is more frequent polling
 - Show last X direct messages
 - Show Xth most recent favorite status for authenticated user
- Authenticated user (Using the Streaming API¹⁵)
 - Show real-time updates from any public (or if allowed, protected) user
 - Display tweets based on keywords or hashtags
 - Display random tweets from the Streaming API's constantly updating stream

¹⁵ The various available APIs and how they can be used is presented in a later section titled “ The Three Available Twitter APIs (REST, Search, Streaming)”

- Display tweets based on geolocation
- Display retweets of a particular chosen tweet
- **Unsupported features (mostly WRITE/POST functions)**
 - Send direct messages to users
 - Post an update to Twitter through the sign
 - Follow new users
 - Get tweets from protected users who have not shared their tweets with the authenticated user

2.6 Basic Hardware Layout

Having determined the requirements for the TweetSign, we had to determine what hardware we would need. Given that we want the TweetSign to operate wirelessly, we have to have some form of embedded computer system with wireless networking that can interface with Twitter and download tweets. The embedded system needs to be connected to a scrolling LED sign so that it can display the information. Since the embedded computer system does not have any way for the user to input information, initial user configuration must be done through a full computer system. This layout is illustrated in Figure 5 below.

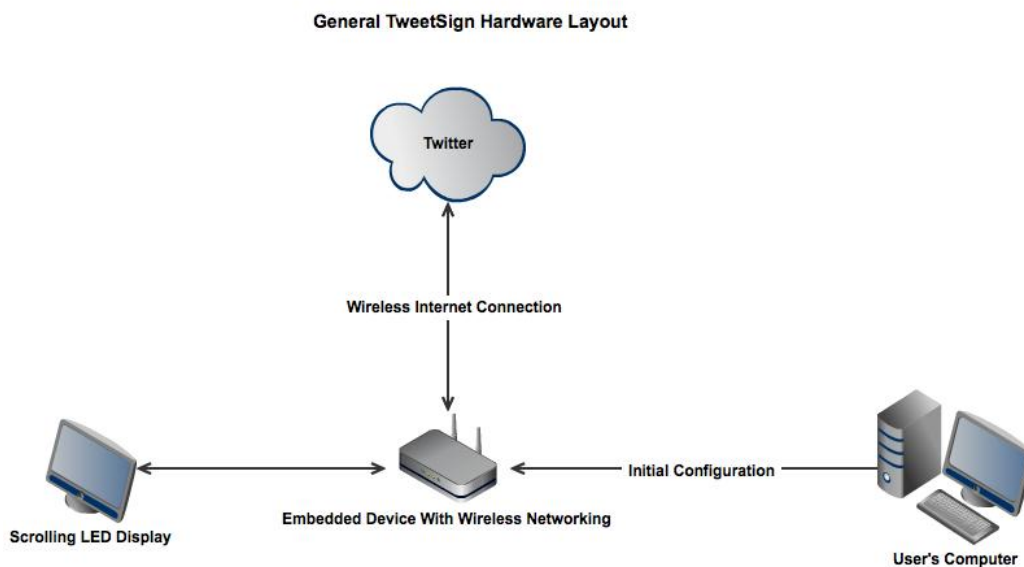


Figure 5: The general hardware layout of the TweetSign.

2.7 Choosing a Scrolling LED Sign

The scrolling LED sign is a crucial part of this project. Ultimately, this is what provides the visual appeal for the potential customers and the core basis for the entire project. Choosing an appropriate sign and vendor would dictate many other product design choices, such as how the code should be written, what will be used to interface with the sign, what cables will be needed to communicate with the sign, etc. With so many choices for signs out there, we selected several factors that were our primary basis for the final decision. The most important of these factors was price. Due to pressure from our primary competitor, we had to choose a sign that was well within \$249.

Therefore, we immediately eliminated any sign that was over \$250. This elimination resulted in many multi-color and multi-line LED signs being thrown out of consideration. A couple of examples can be seen in Figure 6 and Figure 7. An example of a 2-line LED sign for \$378.30 can be seen at KustomXpress¹⁶.



Figure 6: The Sign, Msg, Scroll, 24Chr, Bk LED sign from Amazon for \$404.49 (Source: <http://ecx.images-amazon.com/images/I/21CauNkz2rL. SL500 AA300 .jpg>)



Figure 7: BETAbrite prism indoor color LED display sign - custom quote from Salescaster set the price at \$250. (Source: <http://www.betabrite.com/images/bbprism.jpg>)

On the other end of the spectrum, the ones under \$90 were generally too small (less than 20" in width), had no serial inputs, or were not at all programmable. Some of these LED

¹⁶ <http://www.kustomxpress.com/2P7R-2-Line-4ft-LED-Red-p/2li1616r.htm>

signs were only programmable either through buttons attached to the sign, or a separate remote control. Since we wanted a sign that was not very small, these options were eliminated as well. A list of signs, along with pricing details and reason for elimination are included in “Appendix B – Scrolling LED Sign Table”.

The process of elimination led us to our final choice for the scrolling LED sign: Ultra-Glow-2™ Indoor Brightness Red 26" LED Sign. This product is sold at SignsDirect¹⁷ for \$109.95. This sign uses the MovingSign Communication Protocol v2.1 and while the documentation is not great, we should be able to work with it.

2.8 Contents and Details of the Chosen Scrolling LED Sign

Figure 8 demonstrates the sign within the package and a box which contains the necessary cables and other accessories.



Figure 8: The Ultra-Glow-2™ Indoor Brightness Red 26" LED Sign within the package.

In Figure 9, the contents within the box included with the order of the Ultra-Glow-2™ Indoor Brightness Red 26" LED Sign are shown. A single page of instructions (double sided) has information about setting up quickly to program the LED sign from the computer. The provided mini-CD has drivers that can be installed in order to get the USB-to-RS-232 adapter working properly with the computer. The power cord included is roughly three feet long. Two other

¹⁷ <http://www.signsdirect.com/Home/LED-Signs-Programmable/7x80-LED-Indoor-Brightness-Sign-Red.html>

cables are also provided: an RJ25 connector and a USB-to-RS-232 cable. The unbranded CD comes with pre-built software which allows for programming static messages onto the sign. There is also a remote to turn the LED sign off/on and program it without needing to connect the sign to a computer. Finally, fixtures that can be attached to the sides of the LED sign are included; these are for mounting the LED sign if desired.



Figure 9: The additional contents and the scrolling LED sign. The contents include one page of instructions, a mini-CD with drivers, a regular CD with software, the power adapter, the USB-to-RS-232 adapter, the RJ25 connector, the remote, and the fixtures that can be used to mount the LED sign.

Figure 10 provides a visual to show how the LED sign compares in size to a common object. The dimensions of the sign listed on the website are 26" (wide) X 4.25" (tall) X 1.75". It can be seen from the image that the width of the LED sign is slightly smaller than that of a typical door.



Figure 10: Holding the LED sign in front of a door in order to show relative size.

Once the power source for the LED sign is connected, it automatically starts up in scrolling mode with the message “led display”. Every few seconds, it displays the same message but cycles through a variety of scrolling modes such as roll up and roll down to demonstrate various features of the sign. A static version of the default message when the scrolling LED sign is on can be seen in Figure 11. Using the provided software, the text “TweetSign” was programmed onto the sign. The result is shown in Figure 12.



Figure 11: Default LED message once the power source is connected.



Figure 12: Custom static message programmed on the LED sign using the provided software.

2.9 Choosing an Embedded Platform

Having chosen a scrolling LED sign to display the Tweets, we needed a way to download the Tweets from Twitter. In order to receive this information from Twitter, there needs to be an embedded computer system inside the TweetSign. One of the major design decisions concerned which embedded platform to use. We looked at using a Mini-ITX motherboard, Arduino, and off-the-shelf home router. Each platform has many advantages and disadvantages.

2.9.1 Mini-ITX

Mini-ITX is a compact Personal Computer motherboard form factor. It packs an entire desktop computer system onto a 17 x 17 cm board. A standard Mini-ITX motherboard includes a low power passively cooled processor, onboard video, onboard audio, and onboard Ethernet. Random Access Memory (RAM) and flash or hard disk storage are required to operate the system.

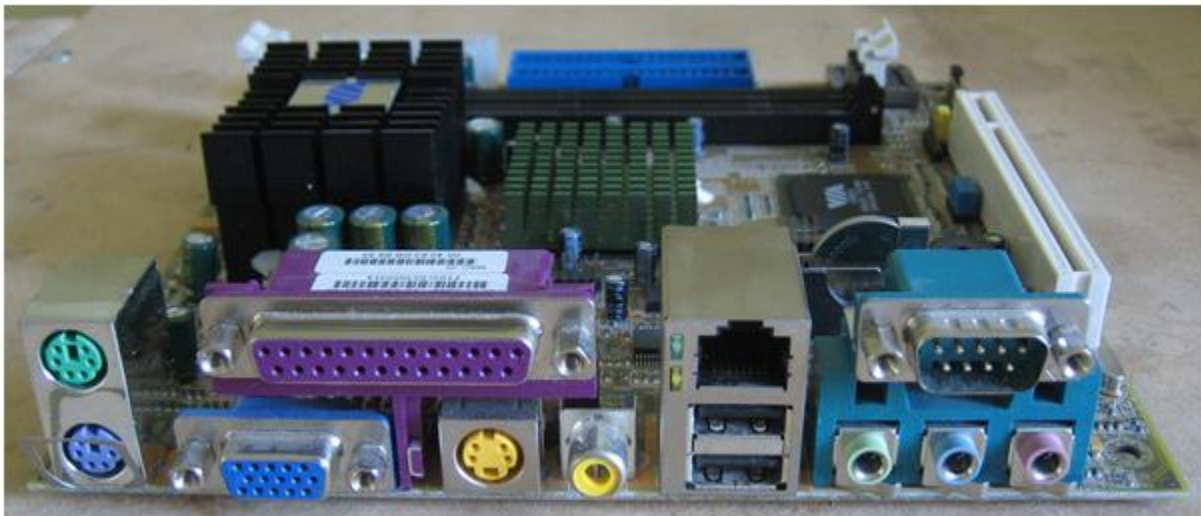


Figure 13: Standard passively cooled Mini-ITX motherboard. (Source: <http://upload.wikimedia.org/wikipedia/commons/2/23/Mini-itx-motherboard.jpg>)

The Mini-ITX motherboard is the most powerful and most expensive option we looked at. This platform has a number of advantages, including speed, ease of programming, and

customizability. However, it also has a number of disadvantages, such as price and power consumption.

The speed of the Mini-ITX system is orders of magnitude greater than any other embedded platforms we looked at. Most Mini-ITX motherboards use dual core Intel Atom processors clocked at 1.8GHz. These are standard x86 architecture processors designed to run modern desktop operating systems. As a result, they are capable of running multiple power-intensive applications simultaneously. Using a Mini-ITX board would allow us to run a stable, lightweight desktop Linux distribution. Given the customizability of Linux, we would be able to code in virtually any language we want to. In addition, resource usage and efficiency would be of minimal concern while programming. As a result, the Mini-ITX system offers the simplest and most streamlined platform for software development.

However, all of these advantages come at the cost of a high price, and relatively high power consumption. A relatively low-end Mini-ITX motherboard costs approximately \$80. By itself, this is not very expensive, but also required to operate the device are a power supply (\$35), RAM (\$15), and hard drive (\$45). Added all together, the Mini-ITX board starts to get prohibitively expensive as an embedded system. The total cost of using a Mini-ITX system would be upwards of \$175. Another drawback of the Mini-ITX is its power consumption. An average Mini-ITX board consumes approximately 20-30 Watts while idle¹⁸. While this is tiny compared to large desktop machines than can consume many hundreds of Watts, it is still power-hungry compared to small microcontrollers that consume mere fractions of 1 Watt. Power consumption is not a large concern for a TweetSign that is plugged into a wall outlet, but it could cause serious problems if we decide to build a battery-powered version.

2.9.2 Arduino

The Arduino is a single-board microcontroller that uses an Atmel AVR processor. The Arduino uses a modular design consisting of a main processor board, and various “shields” that stack on top to provide added functionality. The use of standardized hardware and peripherals

¹⁸ http://www.mini-box.com/site/mb/Power_MB.htm

allows programs to be written at a fairly high level using functions in the Arduino Integrated Development Environment (IDE). This allows for fast development on the Arduino for a wide array of different applications. However, this ease of programming and vast functionality comes at the cost of a high price for the main board and peripheral shields.



Figure 14: Arduino Uno main board. Headers on the edge are for stacking I/O “shields” (Source: http://www.liquidware.com/system/0000/3648/Arduino_Uno_Angle.jpg)

One major advantage of the Arduino is its small size and low power consumption. The Arduino is 2.7 x 2.1 inches, which makes it the smallest embedded platform we looked at. The onboard Atmel AVR processor uses very little power, and can be easily operated on a small battery. The Arduino also offers easy programming using the Arduino IDE. Almost all of the low level programming is abstracted into high-level functions. Even interfacing with the network is easy with Arduino.

The standardized hardware and simple IDE of the Arduino comes at a high price. The Arduino Uno main board costs \$30. This gives the processor and basic USB interface, but no network interface is included. The Ethernet shield costs \$45, and the Wifi shield costs \$50. This gives a total price of \$95 for a Wifi enabled Arduino. For a small microcontroller system, this is

a prohibitively high price. Another major disadvantage of the Arduino is the difficulty of end-user customization. There is no obvious way to take user input to customize the device.

2.9.3 Off-The-Shelf Router

The extremely high price of the Arduino Wifi shield spurred our search for a less expensive Wifi-enabled embedded platform. Home routers are a good example of a low cost embedded system with integrated Wifi. Most routers run a highly customized version of Linux. Some routers that offer easily replaceable firmware can be loaded with a customized firmware that unlocks the full functionality of the router. This allows for running of custom applications that utilize the router's hardware. Repurposing a router allows for a high level of customization on a relatively low cost platform. The main disadvantage of using a router is the limited quantity of flash memory.

OpenWrt is an ultra-lightweight Linux distribution designed to be run on routers. By flashing OpenWrt on a home router, one can install and run custom applications that utilize the hardware in the router. This offers a high degree of flexibility, allowing for programming in high-level interpreted languages like Python. Given that the router is running an operating system, it allows for multiple applications running simultaneously. For example, this means that a router could be running an application that pulls data from twitter, and could also have an apache web server hosting a user configuration page.

The low cost of home routers is what makes them such attractive embedded platforms. Inexpensive routers can cost as little as \$30. Even premium routers with a USB port, 802.11n, and 8MB of flash can be bought for \$80 or less. Even at \$80, this is still less than a Mini-ITX motherboard. And, unlike the Mini-ITX, the router doesn't require any external peripherals to operate.

The main disadvantage of the router is its limited quantity of flash memory. Most home routers have either 4MB or 8MB of flash memory. The entire operating system and any applications must be stored in flash memory. This can seriously limit the number and type of applications that can be installed on a router. Languages that require interpreters to be

installed, like Python, can require 2MB or more of flash memory. When programming applications for a router, it is important to be aware of these resource limitations.

2.9.4 Value Analysis

In order to choose an embedded platform, we performed a value analysis based on several criteria. These criteria were flexibility, ease of programming, end user customization, and price. Flexibility is a score based on the range of features that can be implemented on the platform. For example, we considered whether a platform could simply grab tweets from the server, or whether it can support complex features like a built-in web server. Ease of programming is score based on how easy it will be to develop software for the device. Devices that are capable of running modern interpreted languages like Python would receive a higher score here. End user customization refers to the options that we can provide for the end user to configure and customize their device. Price is a score based on the overall price of implementing the TweetSign on that platform.

Once we had determined the criteria we wanted to measure, we had to weight each criterion based on its importance. Based on our competition analysis, we decided that we need to keep costs as low as possible. As a result, we decided to give price a weight of five, making it the most important criterion. We also determined that flexibility would be extremely important to us. Throughout the course of this project, we expect to encounter unforeseen changes, and may need to add features. For this reason, we think that having a flexible platform that can accommodate these changes would be best, and therefore gave flexibility a weight of four. We decided that maximizing user customizability can only help to improve the marketability of our product, and therefore gave end user customization a weight or three, representing moderate importance. Our least important criterion was ease of programming. Although programming on a difficult platform could increase the difficulty of implementing our project, it shouldn't directly impact the end user. Therefore, we feel that ease of programming is least important of our criteria, and gave it a score of two.

Due to its ability to run modern desktop operating systems, the Mini-ITX motherboard is the most flexible option we looked at, and we therefore gave it a score of five. The Arduino is

the least flexible platform, due to the fact that it does not run an operating system or allow for simple implementation of new features. As a result, the Arduino received a score of two for flexibility. In many ways the router is just as flexible as the Mini-ITX motherboard. It runs a Linux based operating system, and has support thousands of packages for a variety of applications. However, the router option has a limited amount of flash memory, which will limit the number of packages that can be installed at once. Due to these resource limitations, the router lost one point and received a final score of four.

Ease of programming was a fairly simple criterion to score. The Mini-ITX motherboard scored a perfect five again due to its ability to run modern programming languages, and that development can be performed directly on the device. The Arduino received a two because it requires programming to be done in the Arduino IDE using C. The code must then be compiled and flashed on to the device. Again, the router loses one point, because the resource limitations make programming a little harder.

End user customization was a metric we devised to represent the options for user customization we could provide for the user. For example, we knew that our ideal form of user customization would take place in a web browser displaying a configuration page hosted by the TweetSign. Both the Mini-ITX platform and the router are capable of running a lightweight web server to host the configuration page. As a result, the Mini-ITX motherboard and router both received a score of five in this category. The Arduino would require some form of USB customization, which we deemed not to be ideal. This caused the Arduino to receive a score of two in this category.

The final category we looked at was price. We broke down the scoring for this category as follows:

- **5:** less than \$50
- **4:** \$50 - \$75
- **3:** \$75 - \$100
- **2:** \$100 - \$125
- **1:** greater than \$125

The cost of each platform can vary based on the exact specifications, but they can be roughly mapped into the ranges above based on the cost of a fairly average setup. Routers can vary greatly in price, however most basic routers compatible with third party firmware can be acquired online for less than \$50¹⁹. Because of this low price, the router receives a score of five for price. The basic Arduino board only costs \$30, but a third party WiFi shield is required for wireless connectivity. The least expensive Arduino WiFi shields cost \$50, giving the Arduino a total cost of \$90 and a score of three for price. The Mini-ITX system can vary greatly by price, but even the least expensive systems have a total cost of \$175²⁰. This put the Mini-ITX system in the highest category for price, and it received a score of one.

Table 1 shows the score each platform received in each category. By adding the score in each category and multiplying it by its weight, we were able to calculate a weighted score for each hardware option. The router was a clear winner, due to its score of a four or better in every category. Although from a technical standpoint, the Mini-ITX motherboard would be an ideal embedded platform, its high price was the primary factor that led to its loss to the router.

¹⁹ Based on prices of popular “hackable” routers on amazon.com

²⁰ Price breakdown: Intel Motherboard with Dual Core Atom @ 1.8GHz \$82, pico-PSU 90W \$35, 160GB Notebook Hard Drive \$45, 1GB SODIMM RAM \$15, Total Cost: \$177. Prices from www.mini-box.com

Table 1: Value analysis comparing embedded platforms.

Criterion	Flexibility	Ease of Programming	End-User Customization	Price	Weighted Totals
Description	<i>The range of features that can be implemented on the device.</i>	<i>Support for multiple and/or modern programming languages like Python. How easy is updating code?</i>	<i>How easy it would be to let the user customize options such as wifi network ssid and twitter settings.</i>	<i>The overall cost of implementing the device</i>	<i>The weighted total for each platform. (Higher is better)</i>
Weight (1-5)	4	2	3	5	
Mini-ITX Motherboard	5	5	5	1	50
Arduino	2	2	1	3	30
Router	4	4	5	5	64

2.10 Choosing a Router Model

OpenWrt is an open source Linux distribution designed to be run on home routers. OpenWrt differs from other custom firmware because it provides a fully writable file system and package manager. The latest version of OpenWrt is 10.03.1-rc6 “Backfire”. It uses the opkg package manager, a lightweight but powerful package management system. One of the advantages of opkg is its ability to set custom destinations for package installations. This can be utilized to install larger packages to a USB flash drive on routers with a USB port. The latest release of OpenWrt has more than 2000 official packages, allowing it to be customized for most situations. The OpenWrt website includes a wiki with extensive documentation and a support forum for troubleshooting. OpenWrt provides a stable and highly adaptable platform for many embedded applications. More information can be found at <http://wiki.openwrt.org/>.

When choosing a router to use as our embedded platform, we had a number of criteria to consider. The first and most important criterion was compatibility with OpenWrt. The OpenWrt team maintains a table of router models and their compatibility status with various builds of OpenWrt²¹. The list is split into three categories: supported, work in progress, and possible but not being worked on. In order to avoid frustration and compatibility issues, we decided to limit our search to routers on the fairly lengthy list of supported devices.

When looking at each router, we had to consider amount of onboard flash memory. The entire operating system along with any other applications and code would need to fit in the limited quantity of memory. Most inexpensive routers have 4MB of flash memory, and some premium routers have 8MB. Very few routers have more than 8MB of flash. 4MB can be fairly limiting, as a very basic OpenWrt image is 2.5MB by itself²². Larger packages such as a full Python installation can be 2MB alone. Choosing a router with 8MB of flash memory provides a lot more room for additional packages.

Some routers provide USB ports for printer sharing and/or network storage. With OpenWrt, these USB ports can be used for a variety of purposes including additional flash storage, as well as other external peripherals. The ability to use a USB flash drive to expand storage is extremely useful, especially for devices with only 4MB of internal flash. This can provide virtually limitless space for installing packages on the router. The USB port can also be used for interfacing with external peripherals, such as a scrolling LED sign for displaying tweets.

The method by which routers can flash their firmware was important to consider. Some routers only allow changing the firmware through the configuration web page. Incorrect configuration or data corruption can leave the user with access to the router and with no way to flash new firmware. Many routers support flashing firmware using TFTP²³ over Ethernet. This can provide a more reliable way to flash the firmware, even when access to the operating system is lost.

²¹ <http://wiki.OpenWrt.org/toh/start>

²² <http://downloads.OpenWrt.org/backfire/10.03.1-rc5/>

²³ Trivial File Transfer Protocol: A very simple file transfer protocol with no authentication. It is mainly used on internal networks to load boot files. <http://en.wikipedia.org/wiki/Tftp>

Using the above criteria as a guide, we selected the ASUS WL520GU router. The WL520GU is compatible with the latest release of OpenWrt, has a USB port, and is flashable using TFTP. In addition, it contains a “failsafe” system that automatically puts the router into firmware restoration mode if boot fails. The router can also manually be put in firmware restoration mode by holding down a button on the back while powering it on. The only technical disadvantage of the WL520GU is that it only has 4MB of internal flash. However, the USB port with its ability to support USB flash drives can be used to greatly expand the amount of available space for installing packages. The WL520GU also has the advantage of a very low price at \$35²⁴. Finally, the router can be easily disassembled and has a small footprint. This makes it ideal for embedding within another device.



Figure 15: ASUS WL520GU router. (Source: http://ecx.images-amazon.com/images/I/516zEuRj3zL_AA1107.jpg)

2.11 The Three Available Twitter APIs (REST, Search, Streaming)

Having chosen the hardware for the TweetSign, we could start deciding on how to approach the software. For developers that wish to allow users to have a separate medium to

²⁴ \$35 on amazon.com as of October 5, 2011

use Twitter, there is a very detailed procedure for doing so. Twitter has created a set of APIs (Application Programming Interfaces) designed for access to various functions of their website. As stated, “In Twitter's case, [they] provide an API method for just about every feature [that can be seen] on [their] website. Programmers use the Twitter API to make applications, websites, widgets, and other projects that interact with Twitter. Programs talk to the Twitter API over HTTP, the same protocol that [browsers use] to visit and interact with web pages.”²⁵

Twitter provides not one, but three different APIs that are available for use. Each API serves its own purpose and provides different tools for the developers to use. The REST API can make unauthenticated calls and get information about a user; it also provides methods for modifying a user's data with authentication via OAuth²⁶. The Search API is used for making search requests on recent data. The Streaming API provides near-real-time data, but only data that has been received after the application makes a connection with Twitter. In order to retrieve any data from Twitter or to change a user's data on Twitter, a request must be sent to the Twitter servers. Requests can be made in multiple ways: without requiring the application to provide the user's credentials, requiring the user's credentials and sending it to Twitter via HTTP every time a request is made, or allowing the user to give access to the application to make requests on behalf of the user (via OAuth).

The **Representational State Transfer (REST) API** uses HTTP requests to get information about users (unauthenticated) or change information about users (OAuth). To just get information about a public user, no credentials are necessary – requests are based on the IP address of the system requesting information; these requests are known as unauthenticated calls. Unauthenticated users are allowed a maximum of 150 requests per hour maximum, whereas OAuth calls get 350 requests per hour max.²⁷ Data can be requested in a variety of

²⁵ <https://dev.twitter.com/docs/api-faq#api>

²⁶ OAuth provides “a simple way to publish and interact with protected data.” (More info can be found at <http://oauth.net/>) In terms of use with Twitter, it allows users to interact with applications without having to share their passwords.

²⁷ <https://dev.twitter.com/docs/rate-limiting>

formats: XML, JSON, RSS, or Atom. It is up to the application to handle the retrieved data and figure out how to use or display it.

The **Search API** is “designed for products looking to allow a user to query for Twitter content.” It provides data about trends and can get data from the recent tweets. The word “recent” in this case is limited to six to nine days’ worth of tweets. This is a very important distinction to make because the Search API cannot be used to search the entire database of all tweets ever created. Authentication is not involved with the Search API – all queries are client IP based. Twitter has not published the rate limits for the Search API in order to reduce abuse. There are a variety of search operators available, ranging from basic keyword search to advanced filtering and negation search. The full list of examples can be seen at <https://dev.twitter.com/docs/using-search>.

The **Streaming API** “allows high-throughput near-realtime access to various subsets of public and protected Twitter data.” It is the most recent API, going into production in 2010. Essentially, using the streaming API, only data that has been created *after* a connection with Twitter has been established can be retrieved; any data produced prior to the establishment of the connection can’t be retrieved using this API. For example, if a developer wishes to retrieve the last 10 tweets made by a particular user, unless those tweets were created after the developer made a connection with Twitter and was listening for that particular user’s tweets, those 10 tweets cannot be retrieved. The API requires authentication via Oauth for all requests. It currently also supports HTTP Basic Authentication, but this will be deprecated in the near future and therefore should be avoided altogether. It is possible to combine the use of this API with the REST API. The rate limiting for the Streaming API is not as clearly defined. Since a single long-lived HTTP request is established, instead of a limit of X requests/hour, the limiting factor is that each account can only create a single connection. Too many connection requests will cause the client’s IP address to be banned.

2.12 Using Oauth to Connect with Twitter

Some features, such as retrieving tweets from a user's home timeline, require authorization from the user. Twitter currently provides both basic HTTP authentication (which requires passing the user's username and password to Twitter with each call) and Oauth (which provides users with a way to allow applications to access their information without having to give up their passwords to the applications). Due to the fact that basic HTTP authentication has been deprecated²⁸, most of the newer applications connect using Oauth.

However, connecting with Oauth introduces new challenges and additional steps for both developers and end users. Developers must first register their applications with Twitter through the screen shown in Figure 16. After the application is approved (which is instantaneous for applications which request read-only access), developers can view the important details that are necessary to connect using Oauth. A screenshot of the details page can be seen in Figure 17 (the consumer secret has been blacked out for security reasons).

Most Twitter libraries already include the necessary URLs, so the important pieces from this page are the consumer key and the consumer secret. These two items are what uniquely identify one application from another.

²⁸ <https://dev.twitter.com/docs/auth/oauth/faq>

twitter developers Search API Health Blog Discussions Documentation

Application Details

Name: *

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description: *

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

WebSite: *

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL:

Where should we return after successfully authenticating? For [@Anywhere applications](#), only the domain specified in the callback will be used. [OAuth 1.0a](#) applications should explicitly specify their `oauth_callback` URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank.

Figure 16: Creating and registering a new application with Twitter. The screenshot does not show the terms of service or captcha image included on the original page. (Source: <https://dev.twitter.com/apps/new>)

Organization

Information about the organization or company associated with your application. This information is optional.

Organization	None
Organization website	None

OAuth settings

Your application's OAuth settings. Keep the "Consumer secret" a secret. This key should never be human-readable in your application.

Access level	Read-only About the application permission model
Consumer key	A5Yta4VbuCkAMX8WrxESSw
Consumer secret	[REDACTED]
Request token URL	https://api.twitter.com/oauth/request_token
Authorize URL	https://api.twitter.com/oauth/authorize
Access token URL	https://api.twitter.com/oauth/access_token
Callback URL	None

Your access token

It looks like you haven't authorized this application for your own Twitter account yet. For your convenience, we give you the opportunity to create your OAuth access token here, so you can start signing your requests right away. The access token generated will reflect your application's current permission level.

[Create my access token](#)

Figure 17: The details tab of the registered application which shows important information necessary to make requests using OAuth. (Source: <https://dev.twitter.com/apps/XXXXXX/show> - The "XXXXXX" must be replaced with the app number.)

When these two pieces of information (consumer key and consumer secret) are provided to a library that supports OAuth, such as Tweepy²⁹, a unique URL can be generated; when this URL is put into the browser, it directs the user to a specific section of Twitter asking whether or not the user wants to authorize the application to use the account. The page also clearly lists exactly what the application will be able to do (such as read tweets from the user's timeline) and what it will not be allowed to do (such as see the user's Twitter password). An example for the application Seismic Web³⁰ can be seen in Figure 18.



Figure 18: Authorization screen for applications. (Source: <https://support.twitter.com/articles/76052-how-to-connect-and-revoke-third-party-applications>)

If the user is already signed in, the “Authorize app” button can be clicked by the end user to allow the app to use the account. Otherwise, the user will have to first log in before reaching the previous screen. If the developer has provided a callback URL, the user's access token and access token secret (which are used to uniquely identify users) are sent to the callback URL, from which the developer can choose what actions to take (such as store the token in a database on the server).

²⁹ <http://code.google.com/p/tweepy/>

³⁰ <https://seismic.com/seismic-social/desktop/>

From the user's perspective, this is generally a simple process. The user clicks a button to connect with Twitter, authorizes the application to use the user's account, and is redirected back to the main application. The main application can retrieve data from Twitter on behalf of the user, and if granted "write" permissions, can also post to Twitter. When the user decides to use the application again, the developer has the choice of either re-authenticating the user or simply using the token information previously received (if stored somewhere on the server).

If the user goes to the settings page on Twitter and revokes the application's access, even though the application may still have the user's token, it can no longer do anything on behalf of the user.

2.13 Choosing a Programming Language

The next step is to determine which programming language is best to use to interface with the Twitter API. The question of which programming language to use generally depends on the application and the developer. Due to the fact that there are so many different programming languages in the world, it is important to narrow down the viable options. One of our primary goals is to reduce the development time for supporting Twitter functionality. We want to spend more time focusing on creating a good overall product that supports many features instead of getting bogged down by trying to retrieve a single tweet. To this avail, we decided to find out which programming languages have Twitter libraries already created that we could just interface with. Having a library available for use makes the development process easier and provides more time to work on other parts of the project. If a good library is already available, it'll do the heavy lifting of making the HTTP calls, setting the right protocol, requesting the right data format, etc.

Not surprisingly, we found that many programming languages had Twitter libraries³¹:

1. ActionScript/Flash
2. C++

³¹ The exact libraries listed for each programming language can be found at <https://dev.twitter.com/docs/twitter-libraries>

3. Clojure
4. Erlang
5. Java
6. JavaScript
7. .NET
8. Objective-C / Cocoa
9. Perl
10. PHP
11. Python
12. Ruby
13. Scala

Most of these programming languages even had more than one library available. At this point, we needed more elimination criteria. Since we're planning on having the code sit on the router and interface with the LED display, we can eliminate all programming languages that are mostly web-based. Also, we felt it would be more productive to work with a language that we already had prior experience with.

After this level of elimination, we were left with five programming languages: C++, Java, Objective-C, Perl, and Python. Compiled languages often have an advantage in terms of final code size, but from our experience, C++ and Objective-C development takes more time than our other options. From the remaining choices, Python made the most sense since development would be easier and more straightforward. In addition, we could get it installed on the router relatively easily (barring size constraints) and not have to go through a lengthy set up process. Another justification for choosing Python is that we have solid experience using it, so if we run into issues, we can resolve the problems more quickly.

Python does have its own drawbacks, however. For one, it is an interpreted language. The Python package on OpenWrt itself takes up a significant portion of flash memory space, leaving less for actual code. Since it is interpreted, the code is likely to perform more slowly. This may or may not present an issue, depending on how slowly it actually ends up performing.

3 Final Design

This section provides an overview of our design as it was implemented in the completed product.

3.1 High Level Design Overview

The design for the TweetSign is based around the ASUS WL520GU wireless router. The router runs the OpenWrt operating system, and all high level tasks are carried out by the Python interpreter. The user can connect to the TweetSign via a web browser on the local network. A web interface can be used to configure wireless connectivity, as well as tweet display options. The router uses the Twitter API over a wireless internet connection to download tweets based on the user's settings. These tweets are sent over a serial connection to a scrolling LED sign, which displays the tweet. For authenticated access to Twitter, a third party server is necessary. The server acts a middle-man and temporarily stores the access token until the TweetSign can download it. A diagram of this system can be seen in Figure 19.

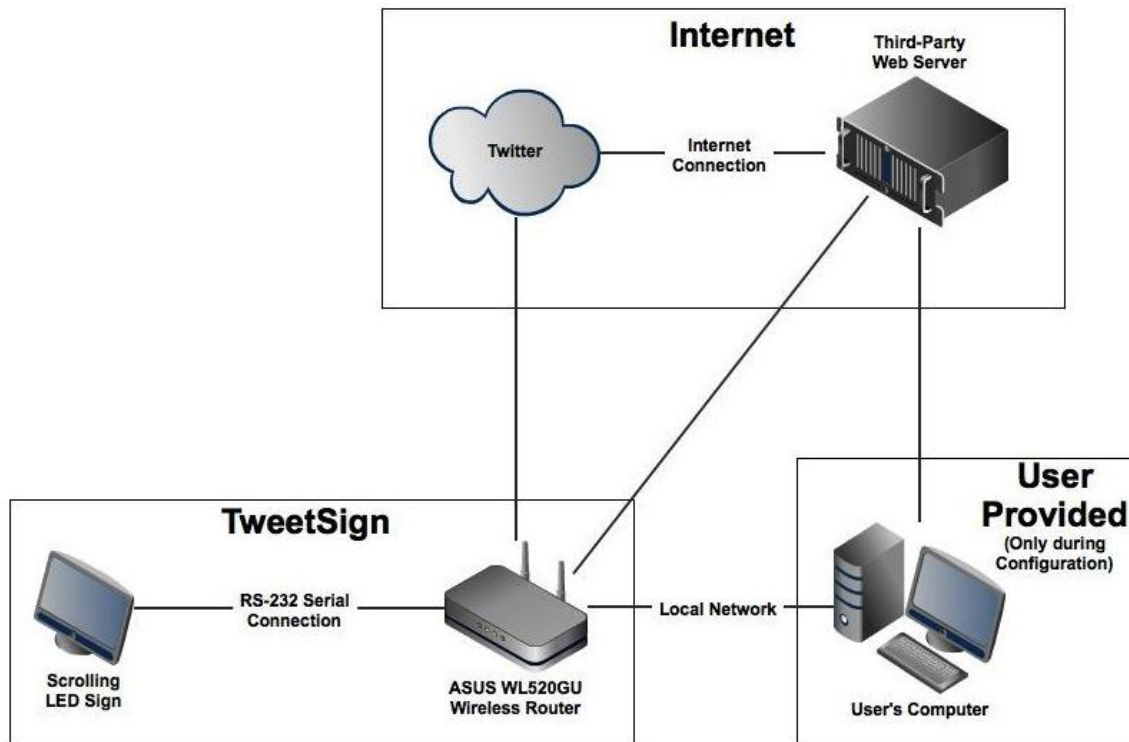


Figure 19: Hardware layout for the TweetSign.

3.2 Features (Operating Modes)

The TweetSign has a variety of features implemented. In the current version, there are six different primary operating modes, of which two have further options. All of the operating modes also support different display modes and scrolling speeds. 24 display modes are supported. However, if the message is longer than the sign can display at once (as will likely be the case most of the time), the display mode does not affect the way the sign scrolls the message. For very short messages, the display modes do change the way they show up on the sign. The supported display modes can be seen below:

- Auto (Random)
- Flash
- Hold
- Interlock
- Roll down
- Roll up
- Roll in
- Roll out
- Roll left
- Roll right
- Rotate
- Slide
- Snow
- Sparkle
- Spray
- Starburst
- Switch
- Twinkle
- Wipe down
- Wipe up
- Wipe in
- Wipe out
- Wipe left
- Wipe right

There are five scrolling speeds. These are defined relatively as normal, slow, slower, faster, and faster. The speed affects all text displayed on the sign, whether or not it can fit within the scrolling area at once.

As far as the operating modes are concerned, the TweetSign can:

- Get and display random tweets from the public timeline
- Get and display tweets from a particular public user
- Get and display tweets from an authorized user's timeline
- Show the current trending topics on Twitter
- Show the daily trending topics on Twitter
- Show static text specified by user

The random tweets mode pulls tweets from all around the world. As a result, many of the tweets are in a foreign language using non-ASCII characters. These tweets will not be displayed on the sign. Private Twitter users cannot have their tweets displayed on the sign unless the TweetSign user who is following those users has authorized TweetSign to access the home timeline. The configuration for accessing the home timeline is more complicated than the other modes and temporarily stores the user's token on the online web server for a short period of time. For the two trending topics modes, the user can further specify whether to include all trends, to show everything but hashtags, or to show only hashtags.

3.3 User Experience

The final model of customization and configuration for the TweetSign is designed to provide a simple and effective user experience. We decided to have configuration occur through a web browser, because this allows the user to change settings while the TweetSign is running. This means that the user can update settings wirelessly from a computer and see their changes reflected while the sign is still running. Although this model provides a great user experience while changing basic settings, it requires the network interface to be completely configured. It is impossible to reach the configuration page if the TweetSign is not yet

connected to a wireless network. Our wireless configuration system is designed to be as simple as possible. Because the network is not configured, the user interfaces with the TweetSign via the USB drive. The user inserts the USB drive into their computer and runs a simple configuration utility. The utility generates configuration files on the USB. The user then inserts the USB drive into the TweetSign and boots it. The TweetSign uses the network configuration files on the USB drive to set up the wireless network. This two-tiered model of user customization is designed to offer the flexibility of a web browser configuration page, while also making the initial network configuration as simple as possible.

4 Methodology

The TweetSign has both a large software and hardware component that must work in tandem to deliver near real-time Twitter updates to the display. This section covers the overall design and logistics of implementing such a system.

4.1 Installing OpenWrt

In order to repurpose the ASUS WL520GU router, OpenWrt was installed on the device. The first step to installing OpenWrt was choosing an image. The image is a snapshot of the operating system that can be copied to the router's flash memory. The OpenWrt website provides a number of precompiled images ready to be flashed.³² These images contain the basic functionality required to boot the router and connect to wired and wireless networks. They come in many different versions that include drivers for specific hardware. For example, the WL520GU is built on a Broadcom 5354 platform. Both the brcm-2.4 and brcm47xx images are compatible with the router³³. The prebuilt images provide a quick and easy way to get OpenWrt up and running quickly. Additional packages can be downloaded and installed by using the opkg package manager. However, this has one major drawback. The package

³² All versions of OpenWrt including prebuilt images and the image generator can be found at <http://downloads.OpenWrt.org/>

³³ For a full list of routers and which target should be used for each, see the Table of Hardware: <http://wiki.OpenWrt.org/toh/start>

manager must first download the package, then extract it and install it. This requires a large amount of free space on the router's flash memory. A far more efficient way to install packages is to include them in a custom image of the operating system.

OpenWrt provides an image generator that can be used to include packages and files pre-installed on the operating system. This is useful because it saves space, as nothing needs to be downloaded or extracted on the router. It can also be useful for preconfiguring wireless connectivity or other configuration files. This can save time when troubleshooting, which often requires frequently reimaging the device. The final image used on the TweetSign was generated using the 10.03-rc6 "backfire" image builder. For a full list of packages and files included in the TweetSign image, see Section 8.4.1.

Before the firmware can be downloaded to the router, it must be put into firmware restoration mode. This can happen automatically if the operating system fails to boot. The router can be manually put into firmware restoration mode by unplugging the router, and holding down the restore button while plugging it back in. The power light slowly blinks to indicate that it is in restoration mode. Downloading firmware to the WL520GU is done via TFTP (Trivial File Transfer Protocol). The router must be connected to a computer using an Ethernet cable. Any TFTP program can be used to download the image onto the router. After the image has been downloaded, the router must be restarted. For full instructions on flashing an image to the router, see Section 8.4.2.

Once the router is running OpenWrt, administrative access can be achieved through a variety of methods. The simplest method of setting up the router is to use a program called luci. Luci is a web user interface that allows basic configuration of the router's core features. Luci is not installed by default; it must either be included in a custom image, or installed manually via command line. Luci is mainly designed for basic access and only allows limited customization. For the purposes of testing, we needed root command line access. Command line access to the router is achieved via unsecured telnet³⁴. The first thing the user is advised to

³⁴ Telnet is a network protocol that gives command line access to a remote machine.

do upon logging into the router is to set a secure password. Once a password has been set, the router can no longer be accessed via telnet, and must instead be accessed via SSH³⁵. Administration via SSH is the only way to get full control over all the features of the OpenWrt operating system.

4.2 Interfacing with Twitter

After the user configures the wireless settings for the router and chooses the mode of operation, Python and PHP code is responsible for talking with Twitter to allow access and retrieve the relevant tweets.

There are two main ways we use to connect with Twitter, both of which require the use of one of the APIs that Twitter provides. For users who wish to display tweets without having to sign into Twitter, unauthenticated calls are made to Twitter for information about public users. This is done through the use of the python-twitter library³⁶ as shown by the 35 lines of sample code in “Appendix C – Python Script communicating with Twitter through REST API”. As mentioned in the background, for these calls, there is an IP-based rate limit.

For our product’s core functionalities, we do not use either the Search API or the Streaming API³⁷. However, the REST API is used extensively throughout the process of interfacing with Twitter. Not only is the REST API used to make unauthenticated calls to retrieve tweets from other users, it is also used to retrieve a user’s home timeline. This way, after being authenticated, users can see the same latest tweet that would show up when they log into Twitter. If they are following private users, the tweets from those users will show up as well.

³⁵ Secure Shell (SSH) is a secure network protocol that gives command line access to a remote machine.

³⁶ <http://code.google.com/p/python-twitter/>

³⁷ The various available APIs were presented in the previous section 2.11 “The Three Available Twitter APIs (REST, Search, Streaming)”

As mentioned in Section 2.12, for most web-based applications, users can connect with Twitter and automatically be redirected back to the application with proper authentication. In our case, because of the nature of the design, we cannot use a callback URL which directly affects the router. Since our main application resides on the TweetSign, we must provide a bridge between Twitter and the TweetSign. In essence, there are four main parts at play for this to work properly. One of them is the python script that is consistently running on the router. It polls the user configuration settings each time it runs and takes the appropriate actions based on what operating mode is currently set. Another is the web server running PHP on the router. This allows the users to control the sign's settings and modes through a web browser. Another very important piece is Twitter itself. All the authorization is done through Twitter's official website and the data is provided by Twitter. The final piece that does the bridging between Twitter and the TweetSign is an online web server that we must have running.

The general idea is that users will follow the instructions on the "advanced settings" page from the PHP that is running on the router. This page warns users that information will be stored on our servers temporarily. From here, they can click a button which reads "Connect with Twitter." The button simply provides a static URL to our online web server (OWS). The files running on the OWS do the heavy lifting. The main page generates a unique request token, uses that token to talk with Twitter to generate a unique authorization URL, and automatically redirects the user over to the TweetSign authorization page on Twitter using our developer credentials and the request token. Every time the main page on the OWS is loaded, a different URL is created. As previously mentioned in the general overview, the URL takes the user to Twitter, where the screen shown in Figure 20 is presented.

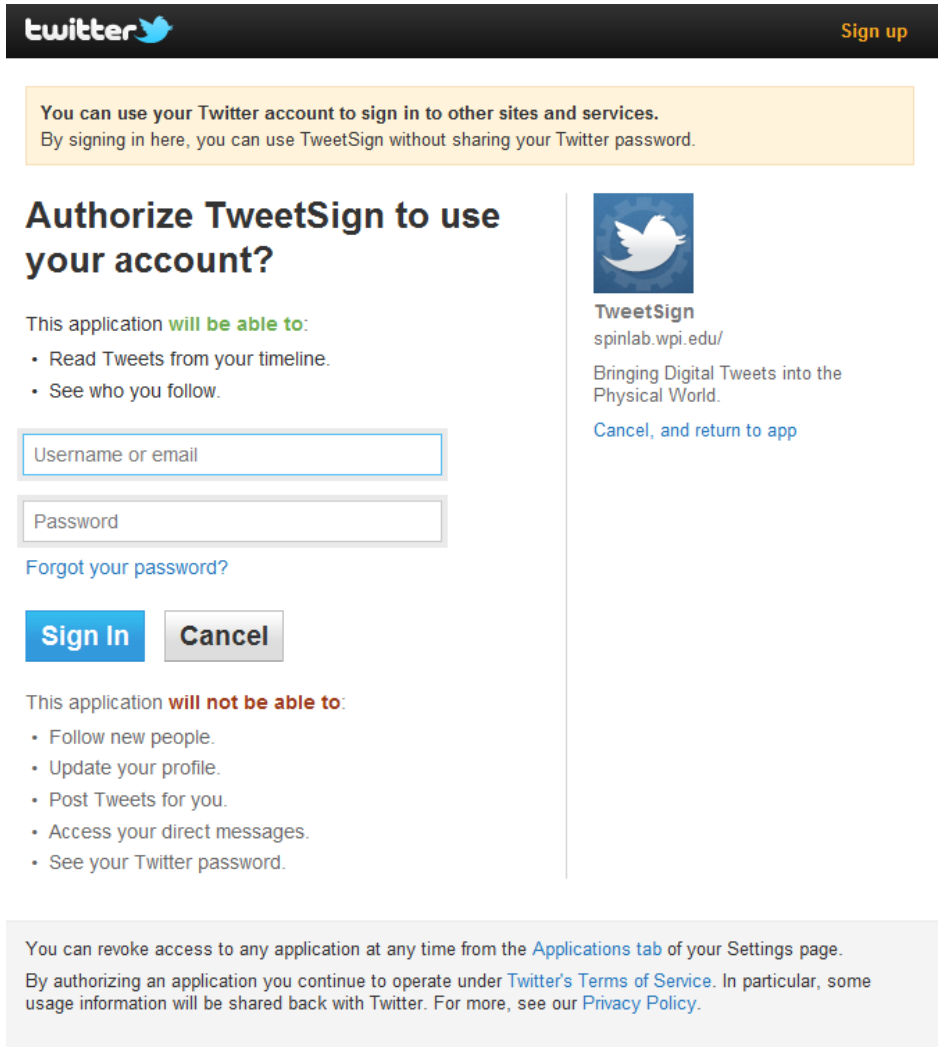


Figure 20: Authorization screen for TweetSign.

If users authorize the application, they are sent back to our OWS with the access token information we need to read tweets from their home timelines. The OWS establishes a validated connection with Twitter using the new information and displays the user's Twitter handle on the page to confirm proper validation. It also generates a random eight digit PIN; using this PIN as a name, it creates a new directory on the OWS and stores the user's access token there. It displays the generated PIN on the resulting screen and prompts the end user to go back to the router's web page.

The web page has an input form field where the user can paste the PIN. Once the submit button is clicked, the PIN gets saved to the TweetSign in a file and the operating mode is set to display the user's home timeline. Since the python script is reading the configuration file after every run, once it picks up the switch to use Oauth mode, it looks specifically for the PIN file on disk. It checks to see whether the PIN is a match to the existing PIN in the config file. When the user first configures Oauth mode, there will be no value in the config file for the PIN. Therefore, the contents of the PIN file will not match the config file; when this happens, the python script will go to the OWS's directory which has the same name as the PIN. It downloads the user's access token and calls a different script on the OWS that takes care of deleting the user's information from the server. It writes the token info to the config file so that it does not need to check the OWS each time (which won't work anyway because the files will have been deleted after downloading it the first time). If the user changes to a different mode and then back to Oauth, there is no need to go through the full process again because the existing PIN will match the PIN in the config file and so python will use the token information from the config file. With the token information available, the script then proceeds to use the Tweepy library to access the user's home timeline. It polls the timeline every so often and if the latest tweet has changed, it will display the tweet on the sign. Otherwise, it will continue polling as usual. A full diagram showing these steps visually can be seen in Figure 21.

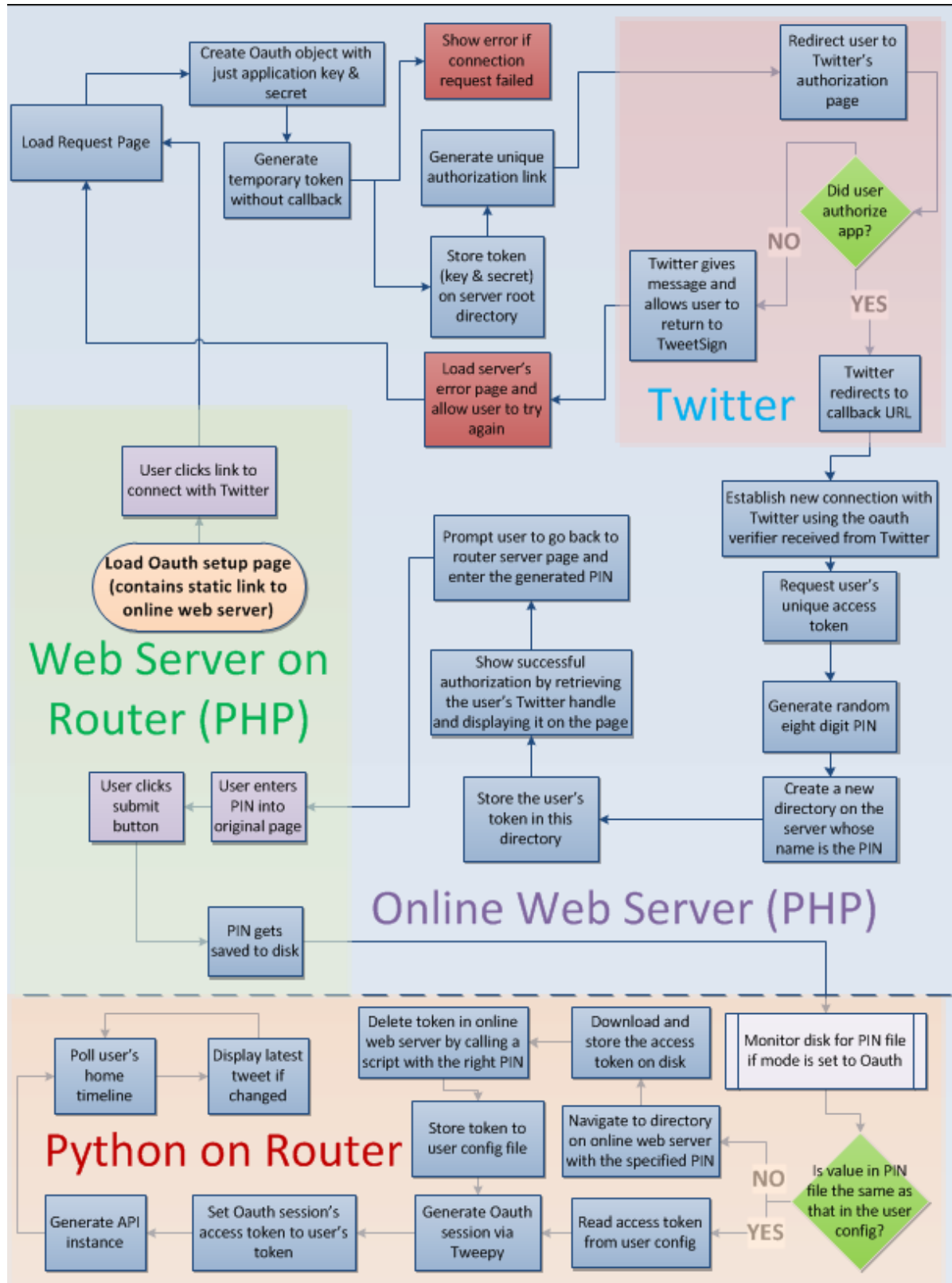


Figure 21: Diagram of how OAuth works on the TweetSign. The starting point is with the setup page (shown in an oval in the left center of the diagram).

4.3 Interfacing with the Scrolling Led Sign

The next step after getting data from Twitter is to display it physically. The scrolling LED sign receives display data on a serial connection from the router. The display uses MovingSign Communication Protocol v2.1. The protocol supports RS-232, RS-485, and Ethernet. However, the Ultra-Glow-2™ sign only supports RS-232, which is what we used for the TweetSign. The serial protocol has a baud rate of 9600BPS and uses 8 data bits with 1 stop bit. It uses three main types of commands: text commands, graphics commands, and control commands. Each of these commands is sent with a data field of variable length containing the actual content to be displayed. The text command can set the text animation, scrolling speed, and display a date and timestamp after the text. It can also set the font and color of the text if supported by the sign. The graphics command can be used to display custom graphics on the display. The control command handles a number of configuration tasks, such as setting the clock.³⁸

Python was used to handle the serial output to the display. In order to physically interface with the serial connection, we used a PL2303 USB to RS232 serial adapter. OpenWrt has drivers for this particular converter, which allowed us to easily access the serial connection using the pycserial Python library.

4.4 Physical Integration

In order to give the TweetSign a polished look, there were a number of physical integration tasks we needed to complete. The first issue was the fact that the scrolling LED sign and the router ran on separate power supplies. In order to make the TweetSign seem like an integrated system, we needed to make them both run off the same power supply. Luckily, both the sign and the router run on 5V supplies, so it was simply a matter of combining the power rails. We decided that it would be best to have the router simply “piggyback” onto the LED sign’s power rails. Our main concern when doing this was whether the LED sign’s power supply would be able to provide enough current to power both the sign and the router. Before

³⁸ Find the full description of the protocol here:
http://www.oceancontrols.com.au/datasheet/hbd/Moving_Sign_Communication_Protocol_%20V2_1.pdf

combining the power supplies, we ran a number of tests to make sure the power supply could handle the load. First, we hooked up an ammeter to the sign, and lit up as many LEDs as would be possible under normal operating circumstances. Next, we hooked up the ammeter to the router, and tested it with all of our code running. When we added the two measurements, we found that it was still far less than the 2A rating on the power supply. This allowed us to proceed with the combining of power supplies. As show in Figure 22, the power supply on the sign has two large terminals with screw connectors. To combine the power supplies, we took the power adapter for the router, cut the cord, and screwed the wires onto the terminals. We were then able to plug the other end into the router. When we power up the sign, power is also supplied to the router through the terminals.

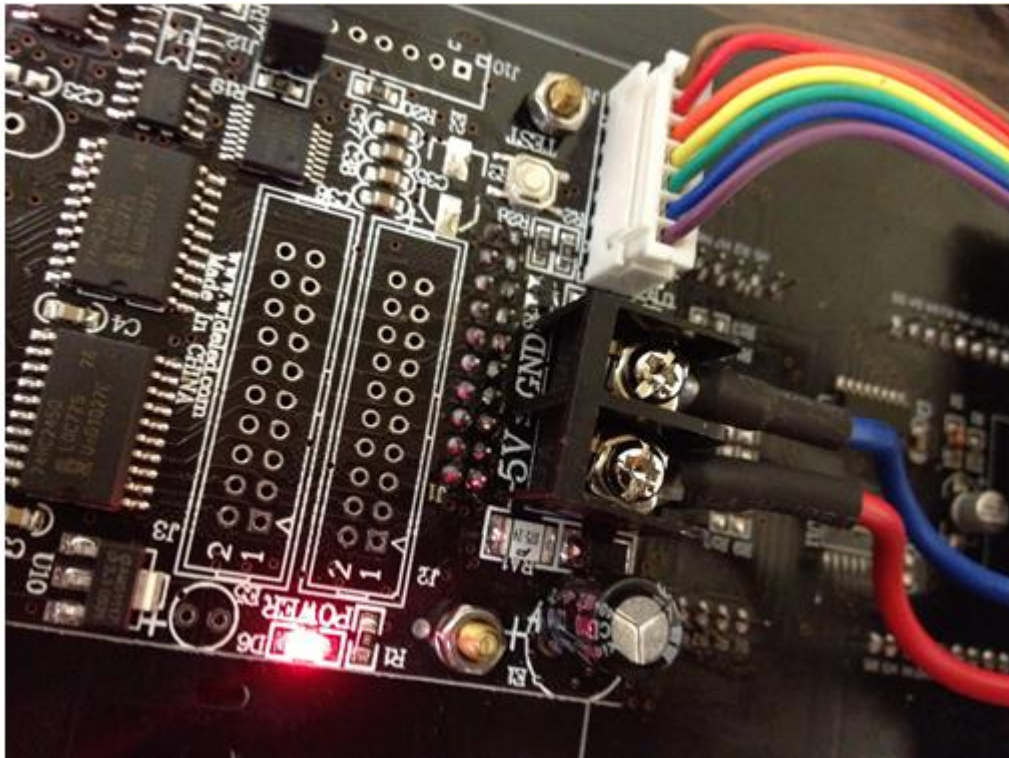


Figure 22: LED Sign Power Rails and Serial Connection

The next aspect we needed to integrate was the serial connection between the router and the LED sign. Our original design had the serial connection going from the router to the external serial input on the sign. This made the TweetSign look unprofessional, so we decided to run the serial cable internally. Figure 23 shows the gray external serial input and the

multicolor ribbon cable that carries the serial signals. To run our internal serial connection, we cut the multicolor ribbon cable in the middle and soldered our serial cable directly to the ribbon cable. This allowed us to remove the external serial cable, giving the TweetSign a more integrated and professional look.

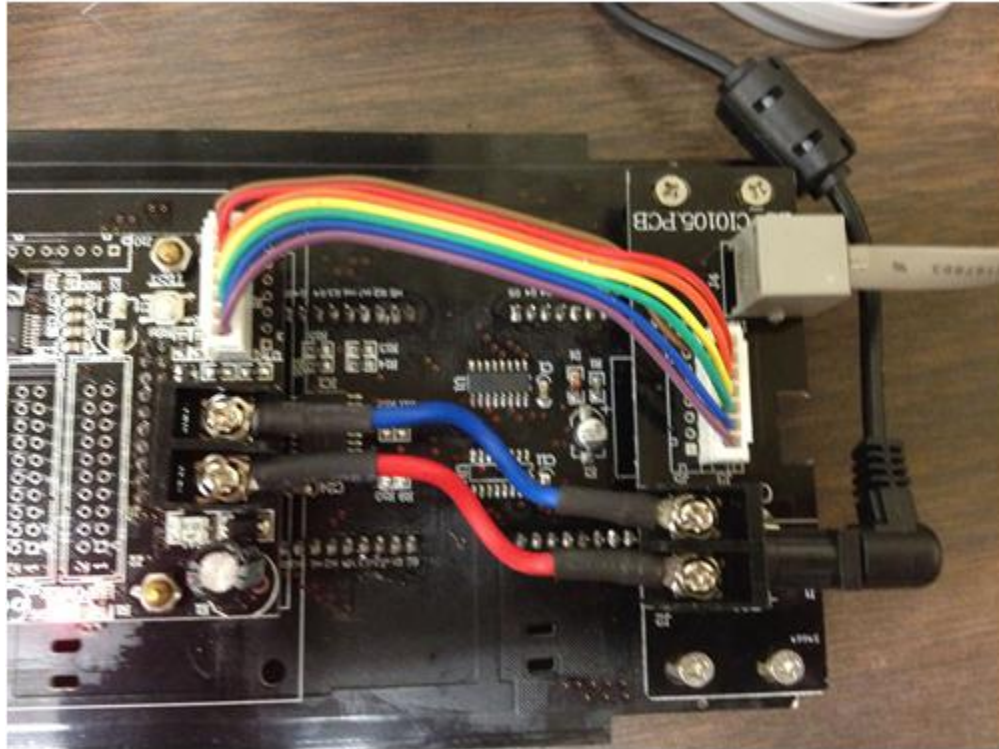


Figure 23: LED Sign Serial Interface

Figure 24 shows all of the cables and adapters that we were able to remove from the TweetSign after integrating the power supplies and serial connection. This drastically cut down on the bulk and cost of the system.



Figure 24: Cables and Adapters Removed From the TweetSign

The final physical integration we made was a case to house the router on the back of the sign. First, we designed the case using SolidWorks. Figure 25 shows a CAD drawing of the case. We then used our CAD file to generate two dimensional images of each piece that could be loaded into a laser cutter. We decided to make the case our of clear acrylic for the first prototype. We chose to use clear acrylic because it is inexpensive, easy to laser cut, and allows the internal electronics to be visible. This is advantageous for demonstrating and explaining how the TweetSign works. Figure 26 shows a side panel of the case being cut in the laser cutter.

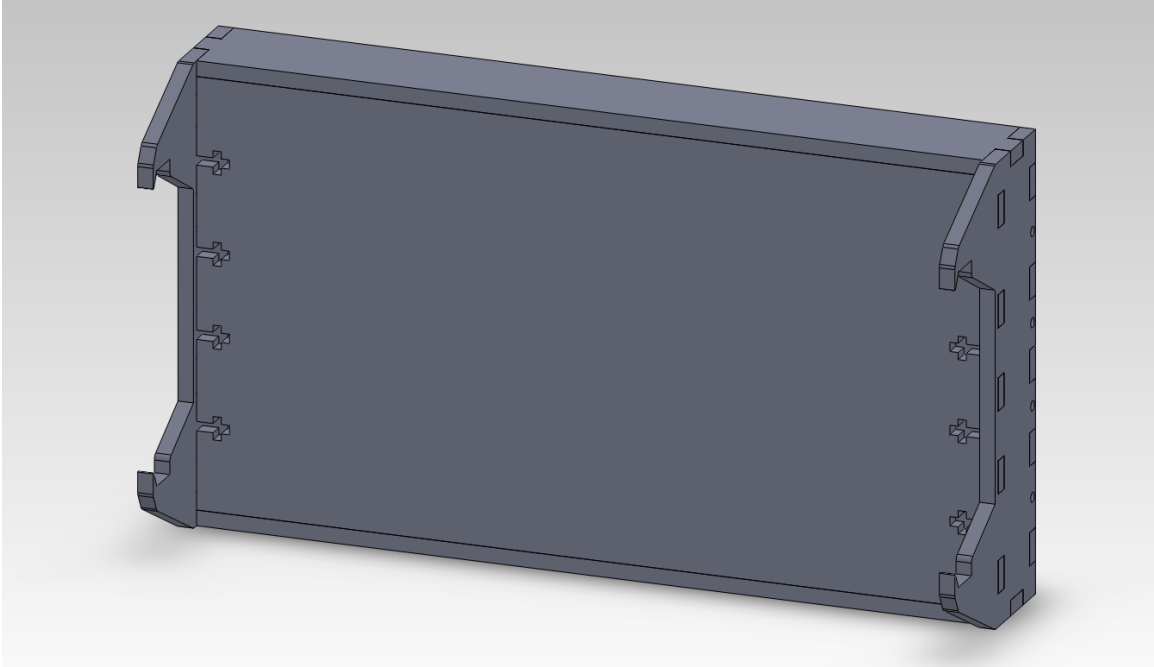


Figure 25: Preliminary CAD Drawing of the Case

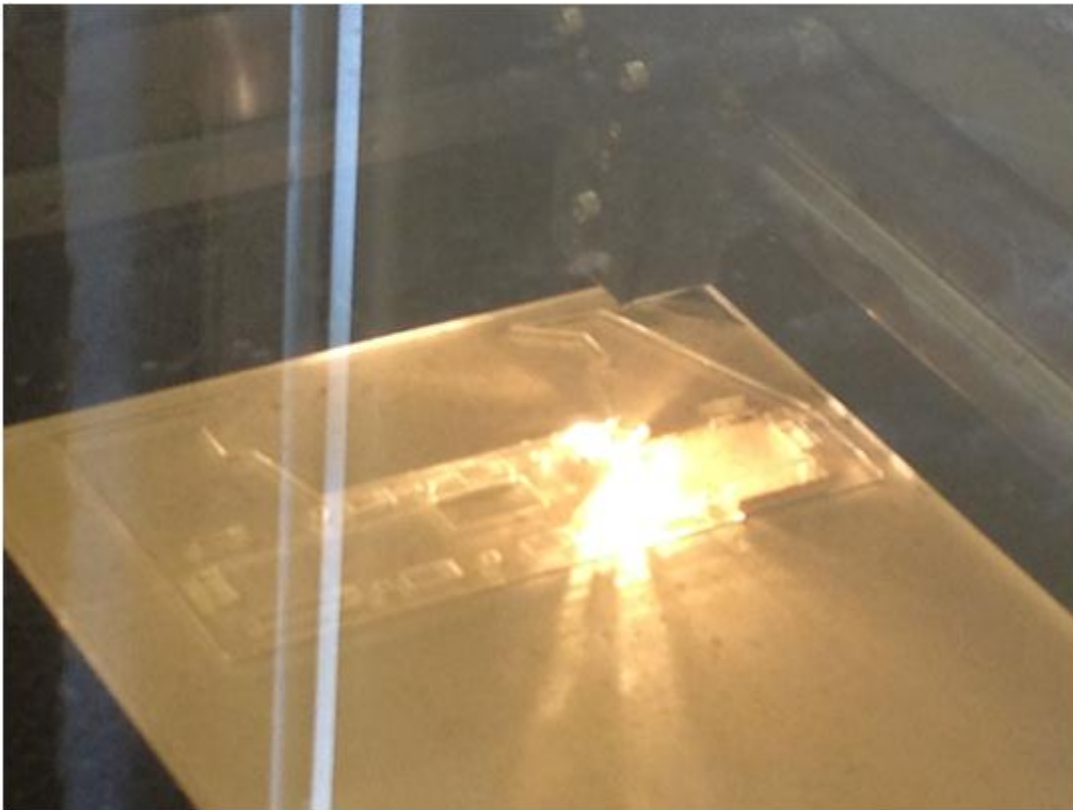


Figure 26: Laser Cutting the Case

4.5 Bill of Materials

Construction of the TweetSign requires the following materials:

- ASUS WL520GU Wireless Router - \$35
- Ultra-Glow-2™ Indoor Brightness Red 26" LED Sign - \$110
- USB Flash Drive - \$5
- USB Hub - \$6
- Acrylic for Casting - \$11
- 16x 6-32 ¾" Zinc Machine Screws w/ Hex Nuts - \$2
- Mounting Tape - \$0.50

4.6 Testing and Debugging

While developing the TweetSign, we went about testing and debugging in a modular fashion. This involved splitting the design into major “blocks” and testing each block individually before putting them together. By testing the internal functionality first, it ensured that each block was functioning correctly, and it allowed for easier debugging of the interface between the blocks. It was also beneficial to test each interface individually before testing the functionality of the entire product.

The high level testing roadmap used during this project can be seen below:

1. Testing Router Functionality
 - a. Ensure that WiFi connectivity is working
 - b. Ensure OpenWRT is running correctly on the router
 - c. Ensure Python is running correctly on the router
 - d. Ensure web server running PHP is working correctly
 - e. Test user configuration page
2. Testing Twitter API Code Functionality
 - a. Test Python Twitter API code on a computer
3. Testing Python – Web Server Integration
 - a. Test PHP user configuration page is able to write properly to the user config file

- b. Test Javascript code correctly changes form fields when different operating modes are selected
 - c. Test Python's ability to read from the config file written to by the PHP
 - d. Test Python's ability to choose the correct operating mode and display the relevant tweets depending on the user config
4. Testing LED Sign Functionality
 - a. Output static text to the sign and make sure it is displaying correctly
 5. Testing Router to Twitter Interface
 - a. Run Python code on the router and make sure it is downloading tweets onto the device and outputting them on the console
 6. Testing Router to LED Sign Interface
 - a. Run LED sign drivers on router and display some test text
 7. Testing Everything Together
 - a. Run everything together and ensure that the TweetSign is downloading tweets and outputting to the display based on the user settings

5 Results

We were able to implement all of the core functionality outlined in the Final Design section, and put the TweetSign in an integrated physical package. The final version of the TweetSign supports five basic unauthenticated operating modes: random tweets, follow single user, current trends, daily trends, and display static text. In addition, it supports one advanced authenticated operating mode which can follow a user's home timeline. Figure 27 shows the completed TweetSign, and Figure 28 shows the acrylic case on the back.



Figure 27: The Completed TweetSign



Figure 28: Back View of the TweetSign

In its current form, the TweetSign has a number of stability and reliability issues in certain operating modes. The system is prone to hangs and crashes while accessing Twitter. This is due to the slow speed and poor optimization of Python on OpenWrt. As an interpreted programming language, it is more processor and space intensive than a compiled language. The router is required to perform a number of intensive tasks simultaneously, including dynamically generating user configuration pages with PHP while interfacing over the Internet with Twitter, and interfacing over serial with the sign. This creates a fairly intense load on the router's resources, and can cause the entire system to lock up and crash. Despite these stability issues while interfacing with twitter, the wireless network connection and interface with the sign are solid, and provide a good embedded platform.

6 Conclusion

The TweetSign is an innovative product that utilizes social media to send real-time updates wirelessly to an LED display from any internet-enabled computer or mobile device. The aspect that sets the TweetSign apart from the competition is the fact that it is a completely self-contained unit that only requires a wireless internet connection to operate. Unlike its competitors, the TweetSign doesn't require a computer or any other external hardware. The final implementation of the TweetSign has a professional aesthetic and provides a variety of operating modes. Although it is prone to minor stability issues when interfacing with Twitter, these issues can be resolved in future revisions.

7 Future Work

In its current form, the TweetSign has a number of limitations. It only supports WPA and WPA2 personal wireless encryption. While the device itself technically supports WEP encryption, the Network Configuration utility does not provide this as an option. WPA Enterprise networks, like the one used on the Worcester Polytechnic Institute campus, are not supported at all. The TweetSign cannot display direct messages sent to an authenticated user. In addition, the TweetSign will only ever display the most recent tweet. Although the 20 most recent tweets are pulled every time (as required by the Twitter API), only the most recent one is displayed on the sign. Although these limitations are minor, they could be fixed in a future version of the TweetSign.

One of the major drawbacks of the current system is the lack of stability and speed. The system tends to work sometimes, but not other times, with nothing having been changed in between trials. This is partially due to the resource limited hardware in the router. One of the biggest performance boosts could be achieved by using a router with at least 8MB of internal flash ROM. This would allow python and its libraries to be installed on internal flash, which is much faster than an external USB flash drive. In addition, a router with more RAM would boost performance. Memory usage on the TweetSign is very high, and having more RAM would allow programs to run more quickly. Another drastic performance increase could come from porting all of the TweetSign code over to C. Python is an interpreted language, requiring a large and slow interpreter to run. If everything were redone in C, the code could be compiled directly to machine code. Not only would this run much faster, but it would eliminate the need to install the large python interpreter. If all of these improvements were made, it would drastically increase the speed and reliability of the TweetSign.

In the TweetSign Pro (term coined by Professor Donald R. Brown), the entire architecture of the system could be very different. Instead of trying to run a web server on the router, having the pages run slowly with limitations on what tasks can be accomplished, and increasing the complexity of the user's experience by needing them to put their browsers to a IP sent out from the sign, we could instead have a fully centralized web server.

Each TweetSign could come pre-shipped with a unique serial number and password that allows the user to create an account on the official TweetSign website. With a centralized server, users could simply create an account on the site, change the settings from anywhere (not just when connected to their networks as is the case currently), and even see previews of what they would see on the TweetSign. Depending on the webhost, an online centralized web server could provide access to a wide variety of options for new features.

For example, the site itself could serve as an application. Users could use the site to visualize tweets, pull tweets based on location, and a whole suite of other tasks. Testing and debugging would be far simpler and stability would depend primarily on the hosting service. Another approach that can be taken with the online web server is to allow it to do all of the heavy lifting, while the router simply polls TweetSign's main server to get only the supported data from some type of storage unit, like a database. This way, the server would be responsible for the implementation of the features while the router only needs to obtain the data provided by the server; making future updates would not require users to download new firmware or change items on the physical TweetSign.

The drawback of this approach is that it potentially introduces a significant cost to maintain servers and push updates to the TweetSign. Too many users requesting data at the same time could cause server crashes, which would most likely upset many users. It also introduces new potential security risks as the server will be accessible worldwide. Privacy issues could also play a role here; in order for the data to be sent from the server down to the sign, the server would have to have knowledge of the data at some point.

One of the most important changes to be made in the TweetSign Pro is better security. Currently, everything except for the authorization by Twitter is unencrypted. The token is pulled from the OWS (online web server) through HTTP GET calls, which could potentially be sniffed. Being able to implement support for SSL would certainly ease some customers' minds.

In terms of future features, there are many available options. One of the most useful features could be to fully utilize the Streaming API. Instead of polling for tweets, we could instead allow the Streaming API to give us near-real-time updates from any public user and

display tweets following specific hashtags. We could also extend the functionality to show tweets for a number of selected users by scrolling through the users and showing the latest tweet for each. For users who are not receiving many tweets, we could add functionality to scroll through the last 10 or so tweets at a predefined interval. Other extensions include showing direct messages, favorite tweets, and tweets from selected lists. Adding the ability to display tweets based on keyword searches (using the Search API) is another potential future feature.

Other future developments could include a wireless configuration setup utility for Windows and mobile applications pertaining to the TweetSign. In addition, the ability to connect to enterprise networks and currently unsupported networks would be available in the TweetSign Pro. Overall, the TweetSign Pro would be a big step up from the currently implemented TweetSign. As a result, its price would also be higher, with the possibility of a monthly or yearly subscription fee.

8 Appendices

8.1 Appendix A – Extended Twitter History and Statistics

It all started back in January of 2006 when Jack Dorsey (@jack) pitched an idea to Odeo, a company based around syndicated content, about SMS-based status updates that would allow a user to let groups of people know what the user was doing. Within weeks, Jack Dorsey, Biz Stone (@biz), and Evan Williams (@ev) created the first version of the Twitter code while working for Odeo. On March 13, 2006 (Jack's birthday), Twtr Beta was created as a way to "stay in touch with your friends all the time." It launched that July. An old screenshot of Twtr can be seen in Figure 29.

Since then, Twitter has a very well-known phenomenon. Users can "tweet" short messages that are limited to 140 characters. Anyone "following" a particular user receives updates about everything the user has posted and can stay informed. Each username is prefixed with an "@" symbol; this symbol makes it very easy to distinguish Twitter usernames (i.e. @jack, @biz, @ev). Politicians started to use Twitter during early 2007 (@barackobama joined in April 2007). In September 2007, when @mtvmoonman started posting tweets from the VMAs, lots of people started paying attention and membership grew rapidly. Celebrities, athletes, and other influential people started using the service and membership continued to increase. On June 12, 2009, "the number of tweets exceeds 2,147,483,647 – the limits of signed integers – causing multiple twitter apps to crash."³⁹

³⁹ <http://visual.ly/history-twitter>

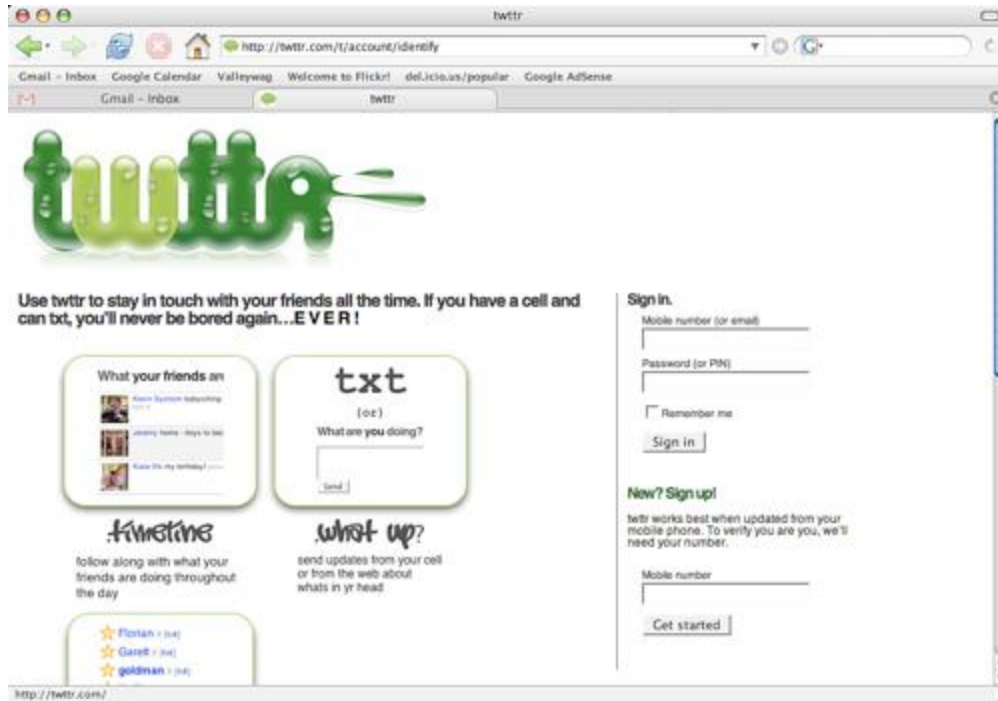


Figure 29: A screenshot of Twtr's homepage in its early days. (Source: http://www.techxav.com/wp-content/uploads/external/farm3.static.flickr.com/2583/3703529021_520dec6533.jpg)

The growth of Twitter in terms of membership count can be seen clearly in Figure 30.



Figure 30: Twitter's adoption rate since its launch in 2006. (Source: The image has been adapted from its original form at <http://mashable.com/2011/09/30/twitter-history-infographic/>)

One recent metric used to measure important historical events and their social impact has been tweets per second. Many users on Twitter will immediately tweet when they hear about something that is relevant to them. People who are following them then become informed and a domino effect takes place. Some of the record-breaking tweet events can be seen in Figure 31.

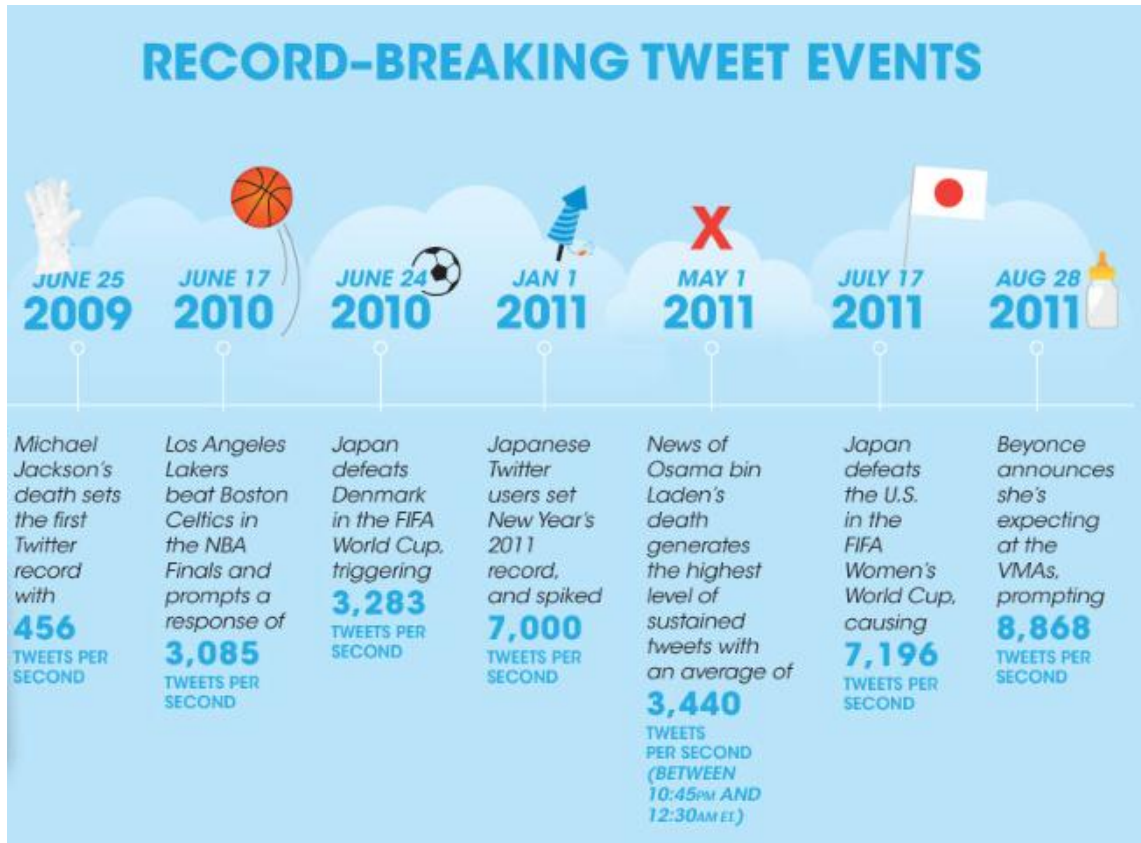


Figure 31: A timeline providing information about select record-breaking tweet events. (Source: The image has been adapted from its original form at <http://mashable.com/2011/09/30/twitter-history-infographic/>)

Some other interesting statistics that support the statement that Twitter has a massive online presence⁴⁰:

- There is a new account created roughly every 5.2 seconds
- Twitter reached its billionth tweet in three years, two months, and one day
- There are roughly 1,650 tweets per second on an average day

⁴⁰ The full set of statistics can be seen at <http://visual.ly/twitter-vs-time>

8.3 Appendix C – Python Script communicating with Twitter through REST API

```
#!/usr/bin/python2.6
# =====
# NOTE:
# FULL CODE AND SET OF FILES CAN BE SEEN AT
# http://spinlab.wpi.edu/projects/tweetsign/tweetsign.html
# =====

def main(argv):

    # Set defaults
    user = 'jack' # Jack Dorsey, Twitter Founder
    tweetToDisplay = 1 # Display latest tweet by default

    # Handle command-line arguments
    confirmationList = ['true', 'y', 'yes']
    for arg in argv:
        # Specify user
        if '-u=' in arg.lower():
            arg = arg.lower().replace('-u=', '')
            user = arg
        # Specify which tweet to display (0 for latest, 18 for oldest available)
        if '-tweet=' in arg.lower():
            arg = arg.lower().replace('-tweet=', '')
            # TODO: Handle this better
            try:
                tweetToDisplay = int(arg)
            except:
                pass

    tweetIndex = tweetToDisplay - 1

    # Start using calls to the python-twitter library
    api = Api()
    statuses = api.GetUserTimeline(user)
    statusesText = [s.text for s in statuses]
    #print statusesText
    print '-'*40
    print 'Tweet From User: '+user
    print "Number of tweets pulled: "+str(len(statusesText))
    print "Showing tweet #: "+str(tweetToDisplay)
    print statusesText[tweetIndex]
    print '-'*40
```

8.4 Appendix D – Comprehensive Guide to Using OpenWrt to Create a Tweetsign

The purpose of this guide is to explain how to use and configure all of the features of OpenWrt utilized by the TweetSign. Most of this information originally came from the comprehensive documentation available at <http://wiki.openwrt.org>. This guide presents it in a format designed to make it easy to replicate and expand upon our work. Please note that this guide assumes a basic knowledge of the Linux command line interface.

8.4.1 OpenWrt Image Builder

The OpenWrt Image Builder was essential to the design process of the TweetSign. Being able to customize an image saves a lot of time when frequently re-flashing the router. Using the image builder requires a computer running Linux.

The image builder can be downloaded from <http://downloads.openwrt.org>. For the TweetSign we used 10.03.1-rc6. This version of the image builder can be downloaded at:

<http://downloads.openwrt.org/backfire/10.03.1-rc6/brcm-2.4/OpenWrt-ImageBuilder-brcm-2.4-for-Linux-i686.tar.bz2>

The image builder archive can be extracted with the following command:

```
tar -xvjf OpenWrt-ImageBuilder-ar71xx-for-Linux-i686.tar.bz2
```

The image builder is actually just a complex makefile that takes a number of parameters. The image is assembled using the make application, which should be installed by default on almost any Linux system. The parameters taken by the makefile are split into three categories:

- **Profile** – The target image
- **Packages** – The packages you want pre-installed on the image
- **Files** – Custom files you want to include in the image

The command to generate the image takes the following form:

```
make image PROFILE= PACKAGES="" FILES=
```

For the TweetSign, we only used the packages parameter, as custom profiles and custom files were not necessary. Almost all of our custom files and scripts were stored on the external USB drive. This gave us more space on the internal flash to install packages. It also meant that our user files were preserved on the USB whenever we re-flashed the router.

The packages included in the standard TweetSign are listed here:

- **kmod-usb-ohci** – required for USB drive
- **kmod-usb-storage** – required for USB drive
- **kmod-fs-vfat** – FAT32 filesystem support for USB drive
- **kmod-usb-serial** – support for usb serial interfaces
- **kmod-usb-serial-pl2303** – support for specific model of serial converter
- **ldconfig** – utility for configuring multiple library locations... useful when installing applications to the external drive
- **libpthread** – dependency for Python
- **zlib** – dependency for Python
- **libffi** – dependency for Python
- **openssh-sftp-server** – not required for release version of TweetSign, but extremely useful for quickly editing and transferring files on the router
- **uhttpd** – web server to host the configuration page
- **php5** – required for dynamic configuration page
- **php5-cgi** – required for dynamic configuration page

Note that Python is not listed here. If Python is included in the image, it will be too large to flash on the ASUS router. Due to these limitations, Python is installed on the external USB drive. Including the packages listed above yields an image file just under 4MB.

The command to generate the image with these packages looks like this:

```
make image PACKAGES="kmod-usb-ohci kmod-usb-storage kmod-fs-vfat  
kmod-usb-serial kmod-usb-serial-pl2303 ldconfig libpthread zlib  
libffi openssh-sftp-server uhttpd php5 php5-cgi"
```

The resulting image files can be found in the bin/brcm-2.4 folder. The image used for the ASUS router will be named openwrt-brcm-2.4-squashfs.trx

For more information about how to use all the features of the OpenWrt image builder, view the wiki article at:

<http://wiki.openwrt.org/doc/howto/obtain.firmware.generate?s%5B%5D=image&s%5B%5D=builder>

8.4.2 Flashing the Image to the Router

This section assumes that you are using the ASUS WL520GU router. In order to flash the router, it needs to be put into firmware restoration mode. This can be achieved by holding down the black restore button on the back while plugging in the router. Continue to hold down the restore button until the power light begins to flash slowly. At this point, connect the router to your computer using an Ethernet cable. The cable can be plugged into LAN port 1-4 on the router, but should not be plugged into the WAN port.

If your computer is running Windows, the router can be easily flashed using the ASUS firmware restoration utility, which can be found in the downloads section at:

http://www.asus.com/Networks/Wireless_Routers/WL520gU/

The process of flashing using the ASUS utility will not be described here, as it is very straightforward and well documented by ASUS.

If you are using Mac OS X or Linux, you will need to use a TFTP client. Mac OS X comes with a command line TFTP client pre-installed. Most Linux distributions should include one as well, but it may vary based on the distribution.

On Mac OS X, open System Preferences and click on the network icon. Temporarily disable wireless (or lower its service order) to prevent conflicts. Select the Ethernet interface and choose “Manually” from the Configure IPv4 dropdown menu. Fill out the fields as show in Figure 32.

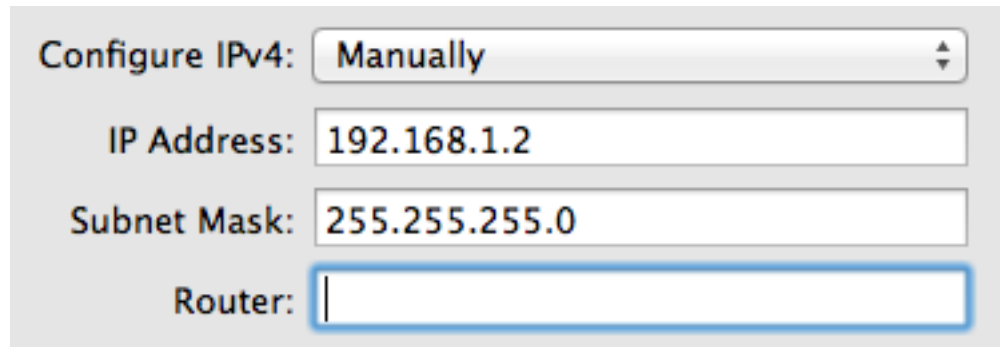
The image shows a screenshot of the Mac OS X network configuration window. The 'Configure IPv4' dropdown menu is set to 'Manually'. Below it, the 'IP Address' field contains '192.168.1.2', the 'Subnet Mask' field contains '255.255.255.0', and the 'Router' field is empty and highlighted with a blue border.

Figure 32: Network configuration in Mac OS X

Open a terminal window (found in Applications->Utilities in finder). Navigate to the directory containing your image file using the `cd` command.

Type the following commands to flash the router:

```
tftp 192.168.1.1  
binary  
trace  
put openwrt-brcm-2.4-squashfs.trx
```

Following the last command, you should see a large stream of output to the console as each block is written to the flash. Once it is finished you can use the command `quit` to exit the TFTP client. After flashing, let the router sit for at least a minute. If it is unplugged directly after the TFTP transfer finishes, OpenWrt will not boot. After it has been sitting for at least a minute, you can unplug and plug back in the power to start up the operating system.

8.4.3 First Run Command Line Access

After the router has been flashed, command line access is achieved using unsecured telnet. If you just flashed the router, make sure to change your network settings back to DHCP so that the router can assign you an IP address. This section assumes that you are using either Mac OS X or Linux. Similar functionality can be achieved in Windows using a program called PuTTY (<http://www.chiark.greenend.org.uk/~sgtatham/putty/>).

Open a Terminal window and issue the following command:

```
telnet 192.168.1.1
```

This should connect via unauthenticated telnet to the router. If it was successful, you should see the OpenWrt welcome banner and command line prompt. It is recommended that the first thing you do is set up a root password so that you can connect securely via SSH. This can be done by typing the command:

```
passwd
```

You will then be prompted to create a new password. After creating your password, you can close the telnet connection by typing `exit`. Now that you have set a password, unsecure login via telnet will not be allowed. Instead you must log in using SSH. This can be done using the command:

```
ssh root@192.168.1.1
```

If it is your first time logging into this installation via SSH, you will be prompted to accept the RSA fingerprint before entering your password. If this is not the first time you have re-flashed the router, or you have previously connected via SSH to a device located at 192.168.1.1, you may get an RSA fingerprint mismatch warning. This is ok, and you can proceed without worrying about it. However, if you are on Mac OS X, the SSH client will not give you the option of continuing past an RSA fingerprint mismatch. To get past this, you will need to clear your list of stored fingerprints. This can be accomplished using the following command:

```
rm -r ~/.ssh
```

If your SSH login is successful, you should be greeted with a similar screen to when you logged in via telnet, except that it is a secure connection.

8.4.4 Network Configuration

This section and future sections require editing of configuration files. This can be done from the command line using the vi editor. The vi editor has a number of complex key commands, but only a few basic ones are needed to edit configuration files. When a file is first opened in the vi editor, it is not in edit mode. You will be able to scroll around the text using the arrow keys, but you will not be able to edit anything. To enter edit mode you can simply press the “i” key. In edit mode, you will be able to edit text normally. When you have finished editing the text, you can exit edit mode again by pressing the escape key. To save the file, type “:w” and hit enter. To exit the editor type “:q” and hit enter. These are the only commands you should need to know to edit configuration files using the vi text editor.

These instructions demonstrate how to configure client mode wireless access. In this mode, the router will connect to the internet through an existing wireless network. This is the main mode that the TweetSign operates in.

Open the wireless configuration file by typing:

```
vi /etc/config/wireless
```

Comment out the following line by putting a “#” in front of it:

```
option disabled 1
```

Change the line:

```
option mode ap
```

To:

```
option mode sta
```

Enter your entire network SSID on the line

```
option ssid OpenWrt
```

If your wireless network does not use encryption, you can save and quit this file now. If your network uses WPA Personal encryption, change the line:

```
option encryption none
```

To:

```
option encryption psk
```

Or, if you are using WPA2 Personal change it to:

```
option encryption psk2
```

Then add a line with your network password

```
option key PASSWORD
```

An example of this configuration file filled out can be seen in Figure 33.

```
config wifi-device wl0
  option type      broadcom
  option channel   11

  # REMOVE THIS LINE TO ENABLE WIFI:
  # option disabled 1

config wifi-iface
  option device     wl0
  option network    lan
  option mode       sta
  option ssid       "Example SSID"
  option encryption psk
  option key        examplepassword|
```

Figure 33: Example wireless configuration file.

You can now save and exit this configuration file. Next open up the network configuration file with the following command:

```
vi /etc/config/network
```

Note that this guide assumes you are using a static IP address, and know some basic information about the network you are connecting to. Scroll to the LAN configuration section and add the following lines:

```
option ipaddr UNIQUEIPADDRESS  
  
option network 255.255.255.0  
  
option gateway ROUTERADDRESS  
  
option dns 8.8.8.8
```

An example configuration file can be seen in Figure 34.

```
#### LAN configuration  
config interface lan  
    option type      bridge  
    option ifname    "eth0.0"  
    option proto     dhcp  
    option ipaddr    192.168.1.5  
    option netmask   255.255.255.0  
    option gateway   192.168.1.1  
    option dns       8.8.8.8
```

Figure 34: LAN section from example network configuration file

You can now save and exit the network configuration file. For the new changes to take effect you must either restart the router or restart the network interface. To restart the network interface, type the following command:

```
/etc/init.d/network restart
```

The router should now connect to the wireless network you specified. If the router successfully connects, you will see the air light on the front of the router illuminate. You can now connect to the router via SSH at the IP address you specified, as long you as your computer is connected to the same wireless network. OpenWrt supports far more networking modes than are described here. For more information, look at the following wiki articles:

[http://wiki.openwrt.org/doc/uci/wireless?s\[\]=wireless&s\[\]=configuration](http://wiki.openwrt.org/doc/uci/wireless?s[]=wireless&s[]=configuration)

[http://wiki.openwrt.org/doc/uci/network?s\[\]=network&s\[\]=configuration](http://wiki.openwrt.org/doc/uci/network?s[]=network&s[]=configuration)

Alternatively you can automatically generate these network configuration files using the TweetSign Network Configuration Utility as described in Appendix E – Getting Started Guide.

8.4.5 Mounting the USB Drive

Before mounting the USB drive, you must create a directory to mount it to. The `/mnt` directory is provided for mounting volumes to. It is recommended that you create a subdirectory for mounting the USB drive. This can be done with the following command:

```
mkdir /mnt/usb
```

The USB drive can be mounted using the following command:

```
mount /dev/scsi/host0/bus0/target0/lun0/part1 /mnt/usb
```

You will now be able to view the contents of the USB drive at `/mnt/usb`.

8.4.6 Installing Packages to the USB Drive

The Opkg package manager in OpenWrt supports installation to external locations like USB drives. This section demonstrates how to install Python to the external USB drive we mounted in the previous section. This method can be used to install any package to the USB drive. On the TweetSign, the following packages are installed to the USB drive: `python`, `pyserial`.

Open the opkg configuration file with the following command

```
vi /etc/opkg.conf
```

Look for the line

```
dest ram /tmp
```

And directly underneath it add the line

```
dest usb /mnt/usb
```

You can add as many custom locations as you want to. In the above example “usb” is the name we are giving to the new destination we are adding, and “/mnt/usb” is the path of location we want to install to. You can now save and exit the opkg configuration file. Before installing any packages, you must update the package list from the OpenWrt repository. This can be done by typing:

```
opkg update
```

Now that the package list is up to date, you can install python by typing:

```
opkg -dest usb install python
```

The identifier “usb” can be replaced with any custom location you create, and “python” can be substituted out for any other package name. If you look on the USB drive, you will see a new folder called `usr`. Inside it are the python binaries and libraries. The directories created on the USB drive like `usr` correspond directly to where the program would normally install on the root filesystem. For example, had you installed Python normally, all the files in `/mnt/usb/usr` would normally be in `/usr`. Python can be run directly from the USB drive with no issues. However, if you are installing libraries to the USB drive, other applications may have trouble locating them. In this case, you will need to create symbolic links in the locations where they should have been installed. For example, say you install a package that creates the file `/mnt/usb/usr/lib/libraryfile.so`. If you had installed this to the root filesystem, it would have been located in `/usr/lib/libraryfile.so`. You can use the following command to create a symbolic link:

```
ln -s /mnt/usb/usr/lib/libraryfile.so /usr/lib/libraryfile.so
```

With the symbolic link created, it will look like the file is actually stored in `/usr/lib`, where other programs expect it to be. However, it is really just a shortcut pointing to the actual file on the USB drive.

8.4.7 Web Server and PHP Configuration

This section outlines how to configure the uhttpd web server with PHP. Use the following commands to create a new instance of the uhttpd web server that will use PHP.

```
uci set uhttpd.llmp=uhttpd
```

This line sets the web server port. This can be changed to any port you want:

```
uci set uhttpd.llmp.listen_http=81
```

This line sets the directory of the webserver. Change this to your web server directory:

```
uci set uhttpd.llmp.home=/mnt/usb/tweetsign
```

```
uci add_list uhttpd.llmp.interpreter=".php=/usr/bin/php -cgi"
```

```
uci set uhttpd.llmp.index_page="index.html index.htm  
default.html default.htm index.php"
```

```
uci commit uhttpd
```

```
sed -i 's,doc_root.*,doc_root = "",g' /etc/php.ini
```

```
sed -i 's,;short_open_tag = Off,short_open_tag = On,g'  
/etc/php.ini
```

You can now restart the web server with the following command:

```
/etc/init.d/uhttpd restart
```

You should now be able to access the web server at the ip address and port you specified. In this example the address would be:

<http://192.168.1.1:81>

8.4.8 Initialization Scripts

Initialization scripts are bash scripts that can be used to initialize processes or perform actions at boot. The structure of an initialization script can be seen in Figure 35.

```
#!/bin/sh /etc/rc.common
# Example script
# Copyright (C) 2007 OpenWrt.org

START=10
STOP=15

start() {
    echo start
    # commands to launch application
}

stop() {
    echo stop
    # commands to kill application
}
```

Figure 35: Basic framework of an initialization script (<http://wiki.openwrt.org/doc/techref/initscripts>)

In the TweetSign, initialization scripts are used to start the various processes to download information from Twitter, and to handle network configuration. Figure 36 shows an example script from the TweetSign that mounts the USB drive and copies over the network configuration files.

```

#!/bin/sh /etc/rc.common
# TweetSign Service Starter
# Mounts USB and copies over network configuration.

START=10
STOP=15

start() {
    # Wait for USB to be available
    sleep 5
    # Mount USB
    mkdir /mnt/usb
    mount /dev/scsi/host0/bus0/target0/lun0/part1 /mnt/usb
    # Copy over config files
    cp /mnt/usb/config/wireless /etc/config/wireless
    cp /mnt/usb/config/network /etc/config/network
    # Restart network interface
    /etc/init.d/network restart
}

stop() {
    # Unmount USB
    umount /mnt/usb
}

```

Figure 36: Example initialization script from the TweetSign

Many of the commands in the above script should be familiar from previous sections of this guide. Commands in the `start()` section of the script are executed when the process is started, and commands in the `stop()` section are executed when it is stopped. When you create an initialization script, it will initially only be a text file. In order to make it executable, you must execute the following command:

```
chmod +x scriptname
```

In order to get your script to run at boot, it must be copied to the `/etc/init.d` folder. This can be accomplished with a simple copy command as shown below:

```
cp scriptname /etc/init.d/scriptname
```

The final step in getting your script to run at boot is enabling it. This can be done with a simple command:

```
/etc/init.d/scriptname enable
```

Now your script should run every time at boot. For more information about initialization scripts, see the following link:

<http://wiki.openwrt.org/doc/techref/initscripts>

8.4.9 Further Reading

All of the information in this document can be found at <http://wiki.openwrt.org>. For convenience, we have provided links to pages that you will most commonly need while dealing with the TweetSign:

- Image Builder: [http://wiki.openwrt.org/doc/howto/obtain.firmware.generate?s\[\]=image&s\[\]=builder](http://wiki.openwrt.org/doc/howto/obtain.firmware.generate?s[]=image&s[]=builder)
- Opkg Package Manager: [http://wiki.openwrt.org/doc/techref/opkg?s\[\]=opkg](http://wiki.openwrt.org/doc/techref/opkg?s[]=opkg)
- Network Configuration: [http://wiki.openwrt.org/doc/uci/network?s\[\]=network&s\[\]=configuration](http://wiki.openwrt.org/doc/uci/network?s[]=network&s[]=configuration)
- Wireless Configuration: [http://wiki.openwrt.org/doc/uci/wireless?s\[\]=wireless&s\[\]=configuration](http://wiki.openwrt.org/doc/uci/wireless?s[]=wireless&s[]=configuration)
- Initialization Scripts: <http://wiki.openwrt.org/doc/techref/initscripts>
- Setting Up a LAMP Stack: [http://wiki.openwrt.org/doc/howto/lamp?s\[\]=lamp](http://wiki.openwrt.org/doc/howto/lamp?s[]=lamp)

8.5 Appendix E – Getting Started Guide

This guide is designed to give you a quick overview of how to set up and configure your TweetSign for use over a wireless network. Before you get started, you will need a wireless internet connection and a computer running Mac OS X. Note that Mac OS is only required if using the TweetSign Network Configuration Utility.

Take the TweetSign USB drive and insert it into a free USB port on your computer. Open a Finder window and view the contents of the drive, as shown in Figure 37.

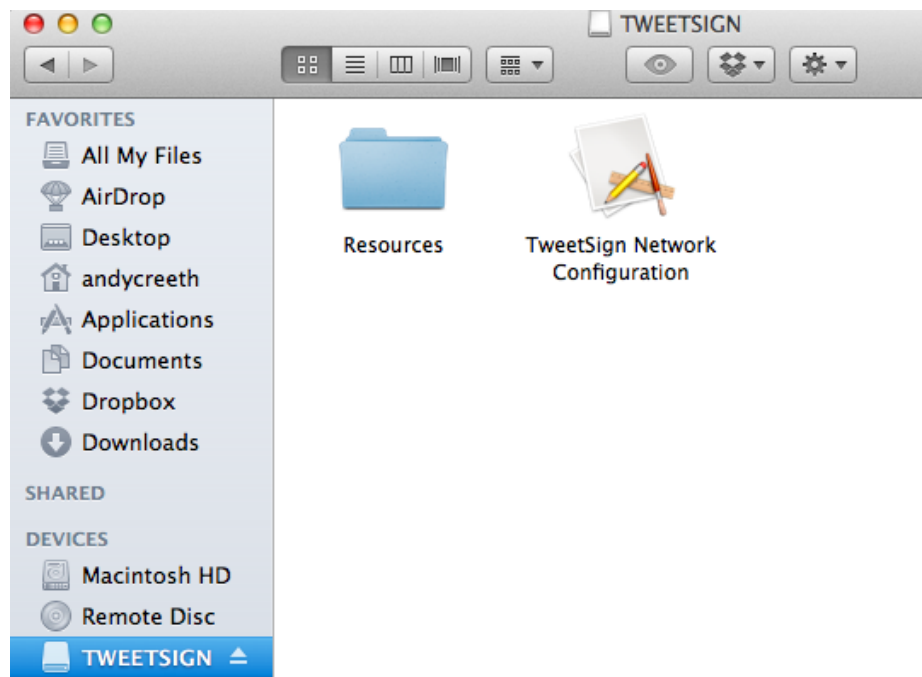


Figure 37: TweetSign USB contents.

Double click the TweetSign Network Configuration icon. The Configuration Utility will launch, and you will be prompted to type in the name of your wireless network, along with the encryption type and password. The TweetSign Configuration Utility is shown in Figure 38.

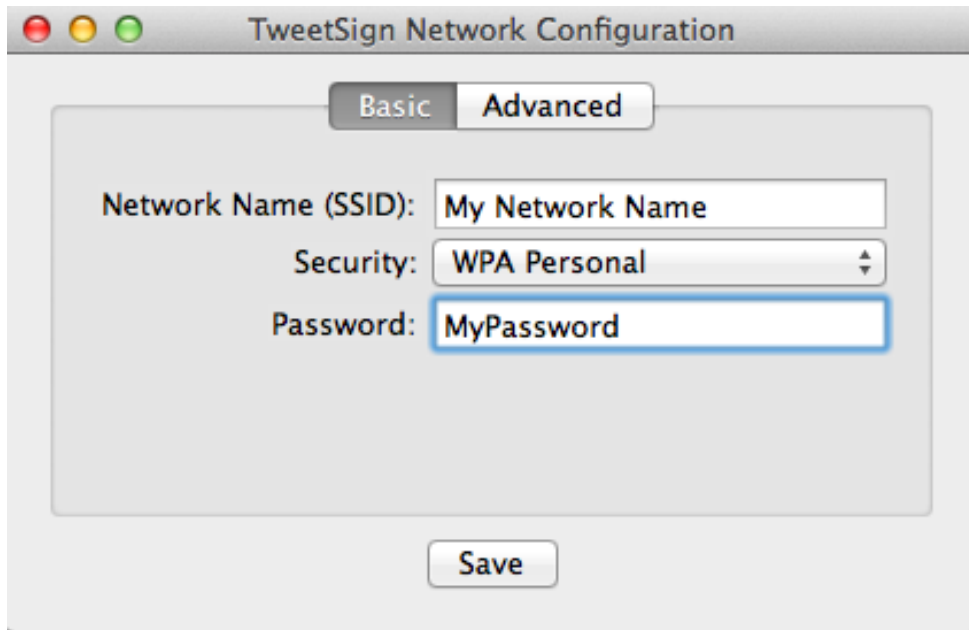


Figure 38: TweetSign Network Configuration Utility

Once you have entered your network information, click the save button. You will be shown a message indicating whether the configuration was successfully saved, as shown in Figure 39.

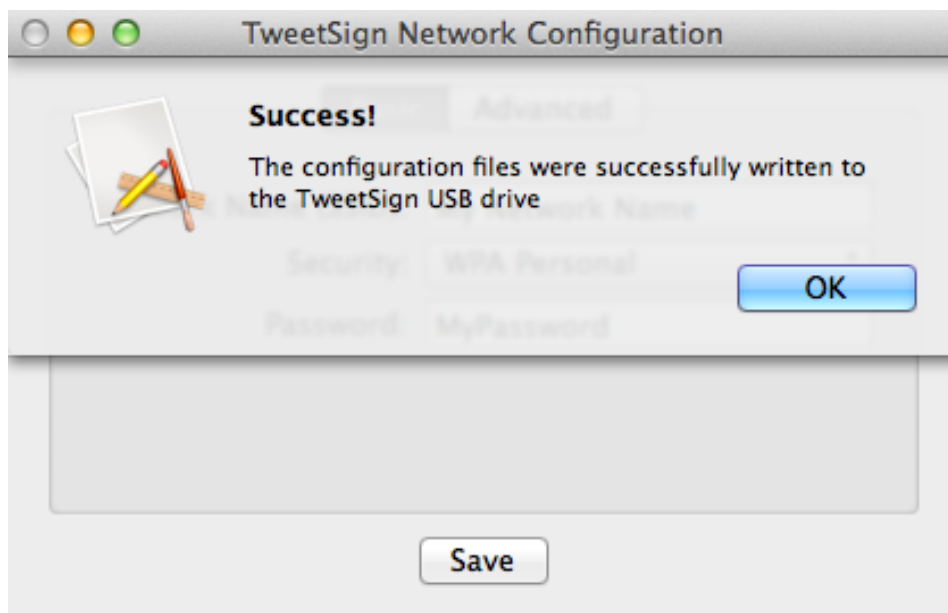


Figure 39: Successful Network Configuration

Now that the network configuration has been saved to the USB drive, you can eject it and plug it into the side of your TweetSign. Once the USB key has been plugged into the

Tweetisign, connect power to the side of the TweetSign. Please allow approximately one minute for the TweetSign to initialize. Once the sign is initialized, you will hear a “beep” and the IP address of the TweetSign will appear on the scrolling LED display. Open up Safari, or another web browser. In the URL field at the top, type in the text as shown in Figure 40, replacing the part underlined in red with the IP address shown on the sign. Hit enter.

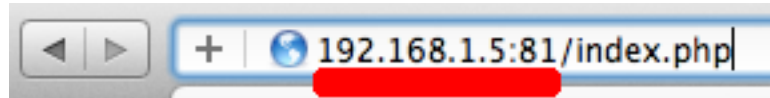


Figure 40: Network Configuration Page URL

Once the page loads, you should be greeted with the TweetSign homepage, as shown in Figure 41. Click the “Setup” button on the top, underlined in red in the picture.

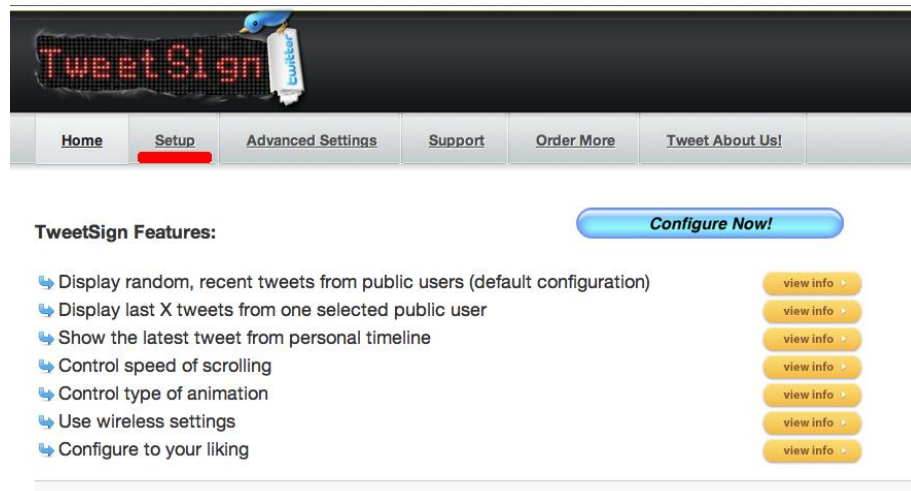


Figure 41: Configuration Homepage

Figure 42 shows the basic setup page. From this page you can configure each operating mode. By default, the sign will show random tweets from random users. You can select a different mode from the Basic Operating Mode drop down menu. Some modes require more information. When this is the case, the new relevant fields will automatically appear.

When you are done choosing the operating mode and selecting the relevant options, click save changes at the bottom of the page. Please allow at least 20 seconds for the page to reload and acknowledge that your changes will save. Within one minute the TweetSign will update to reflect your changes.

Configure Your Options

Current Operating Mode: Displaying recent tweets from random users

➔ **General Settings:** Change these to get set up very quickly...

Basic Operating Mode:
Check "Advanced Settings" for more modes

Scrolling Speed:
How quickly the sign should scroll

Scrolling Mode:
How to display the text on the TweetSign

Figure 42: Basic Setup Page

In order to view your home timeline, you will need to move to the “Advanced Settings” tab. First, make sure you are on the proper page; look at Figure 43 to make sure it looks similar to the page you see.

Current Operating Mode: Displaying recent tweets from random users

➔ **Advanced Settings:** Follow the two easy steps below to set up...

Step 1:
Click the link to the right and authorize TweetSign

Step 2:
Enter Your PIN Received From Clicking the Link From Step 1

Scrolling Speed:
How quickly the sign should scroll

Scrolling Mode:
How to display the text on the TweetSign

Figure 43: Advanced Settings Page

Before proceeding, you must be aware that your access token will be temporarily stored on our server. The token will be deleted immediately after being downloaded to your sign. However, if someone were to obtain your access token, they may be able to read messages from your home timeline. They **will not** be able to post messages on your behalf.

Once you are ready, click on the button next to step 1 that reads “Connect with Twitter”. It will take you to a page that looks similar to Figure 44. If you are not signed in yet, please sign in to authorize the TweetSign to pull tweets from your home timeline. If you are already signed in, you can simply click the “Authorize App” button (or the “Sign In” button in some cases).

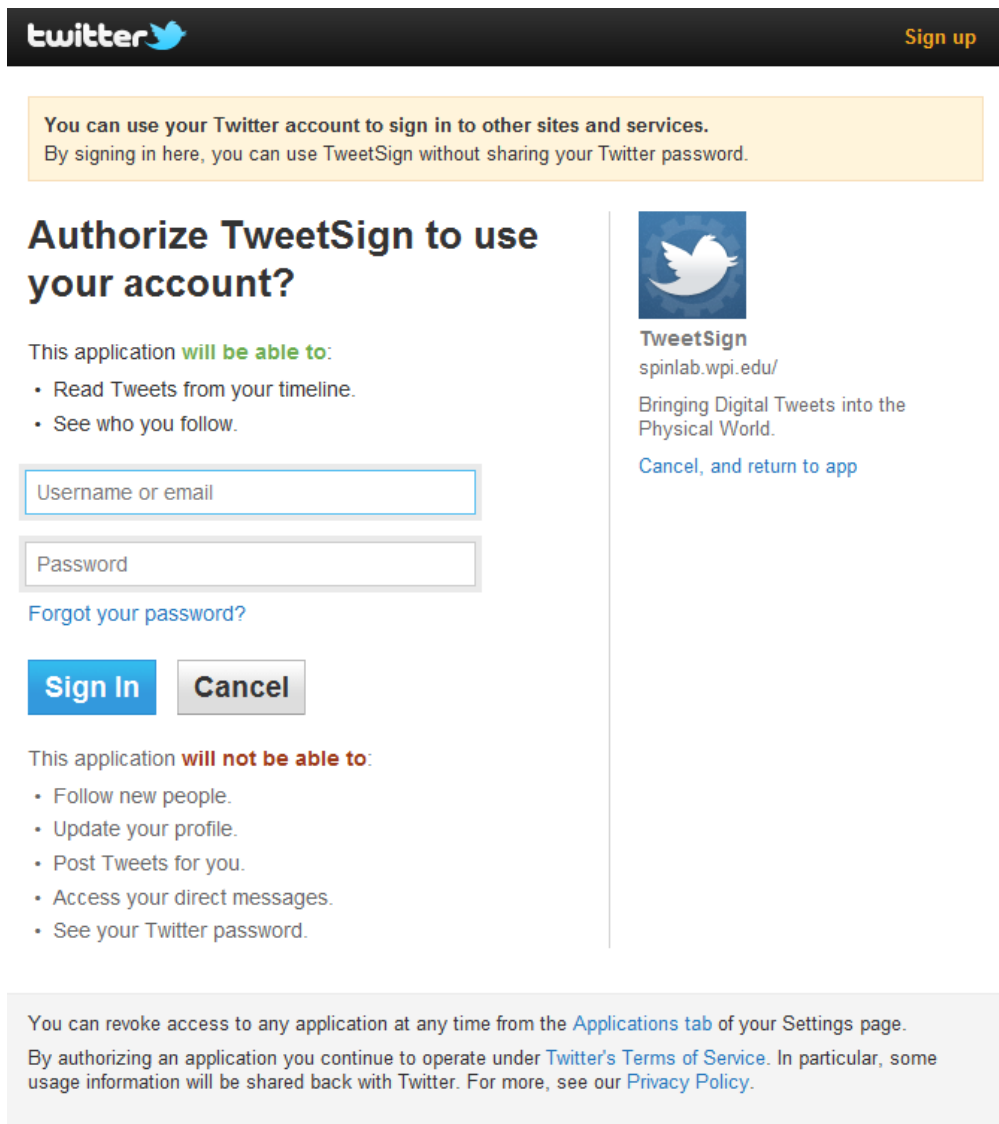


Figure 44: Twitter authorization page for TweetSign

You will automatically be redirected a page that confirms successful authorization. It should look similar to that of Figure 45. An eight digit number in large text will be shown on that page. Please copy that number, go back to the “Advanced Settings” page you were on

earlier, and paste it into the input field for step 2. You may change the scrolling mode and speed if you wish. Click the “Save Changes” button at the bottom. If you copied the PIN incorrectly, you will get an error message. Try pasting it again. If the steps were followed properly, you should see a success message at the top of the page.

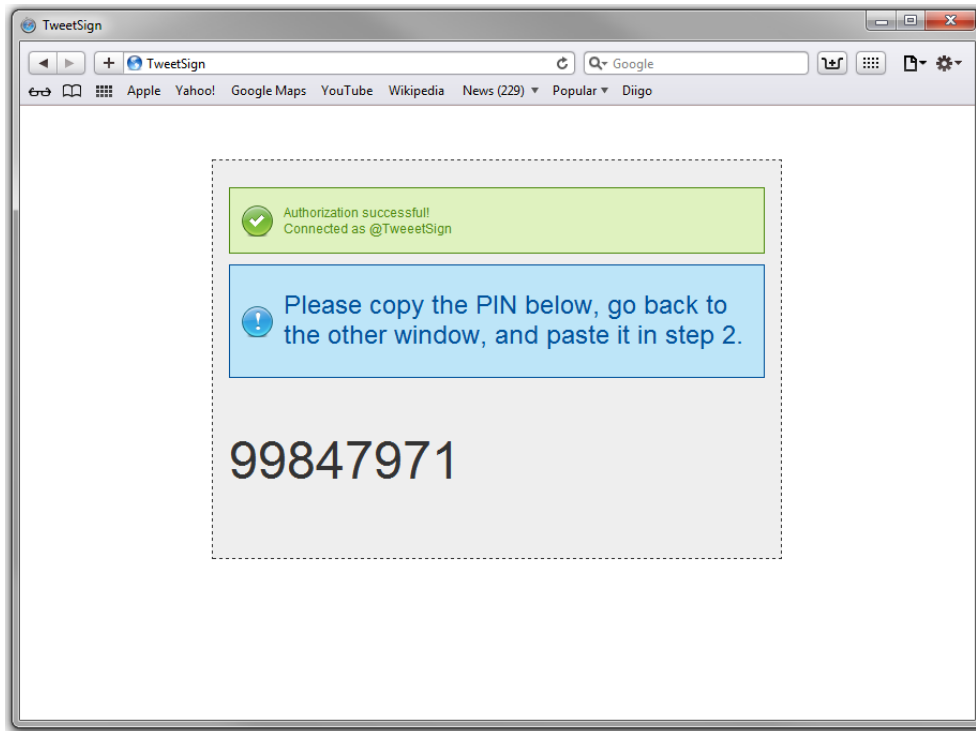


Figure 45: TweetSign Oauth initial success page

You must wait roughly five minutes before the changes can take place on the TweetSign. You only have to go through this process once. If you switch modes (to say, “random tweets”) later and decide to switch back to Oauth mode, do not try connecting with Twitter again and do not change the value in the PIN. It will automatically recognize you if you switch modes correctly.

In the event that you do accidentally click “Connect with Twitter” again, just follow through all of the instructions again to make sure that the sign works as expected.

If the sign does not work properly after 10 minutes, please unplug the TweetSign and plug it back in.