Christopher Peri, Ph.D.

Contribution by Bess Ho

Includes Twitter Development for iPhone and Android

Sams Teach Yourself

the Twitter API

in 24 Hours

SAMS

Christopher Peri Ph.D.

Sams Teach Yourself the Twitter API in 24 Hours

SAMS 800 East 96th Street, Indianapolis, Indiana 46240 USA

Sams Teach Yourself the Twitter API in 24 Hours

ISBN-13: 978-0-672-33110-7
ISBN-10: 0-672-33110-1

Library of Congress Cataloging-in-Publication Data

Peri, Christopher A., 1964-
Sams teach yourself the Twitter API in 24 hours / Christopher A. Peri, Bess P. Ho.
p. cm.
Includes index.
ISBN-13: 978-0-672-33110-7 (pbk. : alk. paper)
ISBN-10: 0-672-33110-1 (pbk. : alk. paper)
1. Application program interfaces (Computer software) 2. Twitter. I. Ho, Bess P., 1967- II. Title. III. Title: Teach yourself the Twitter API in 24 hours.
QA76.76.A63P47 2011
006.7'54—dc23

2011022576

Printed in the United States of America

First Printing June 2011

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Bulk Sales

Sams Publishing offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales. For more information, please contact

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact

International Sales
international@pearson.com

Associate Publisher
Mark Taub

Signing Editor
Trina MacDonald

Development Editor
Songlin Qiu

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Copy Editor
Barbara Hacha

Indexer
Erika Millen

Proofreader
Sarah Kearns

Technical Editors
Doug Jones
Ronan Schwartz
Ben Schupak

Publishing Coordinator
Olivia Basegio

Cover Designer
Gary Adair

Composition
Gloria Schurick

Contents at a Glance

Table of Contents

About the Author

Dr. Christopher Peri received his Doctorate from the University of California, Berkeley, in Architecture. His focus was on Collaboration in Virtual Environments delving into methods that facilitate designers and engineers to improve communication over remote networks.

He started playing with the Twitter API very early in the API release, creating his own Twitter client called TwittFilter, which is geared more to the occasional user then someone who uses Twitter all the time. As time went on, he added more and more features and functions for his own personal use, until one day he realized he had a fairly sophisticated application and opened it up to the general public to use. He learned quite a bit about the Twitter API the hard way—by simply coding things up and seeing what happens. Although TwittFilter is still a personal project, he has already created a number of private Twitter applications, robots, and smaller projects like NewsSnacker.com, which is open to the public.

About the Contributing Author

Bess Ho is a UI Engineer in mobile, tablet, TV, and web with a strong background in data analytic and consumer behavior. She received her Master Degree from the University of California, Davis in Food Science and Technology. Her focus was on Consumer Sensory Science and Engineering. She is the winner of Nokia Open Screen Project Fund and was elected as Samsung Star in the Samsung Mobile Innovator worldwide program. She served as technical editor for the book titled **Building OpenSocial Apps: A Field Guide to Working with MySpace Platform** (Addison Wesley, 2009). She has presented mobile technology at Stanford University, O'Reilly Web20 Expo SF, Where20 Conference, Silicon Valley China Wireless Conference, and many developer events. Currently, she is Mobile Architect (EIR) for Archimedes Ventures. She also advises many early-stage startups in UI/UXP design and mobile development in multiple platforms. She is actively teaching many mobile classes such as iOS SDK in Silicon Valley and online courses at Udemy.com. You can follow her at Twitter @Bess or Slideshare at www.slideshare.net/bess.ho. Her developer blog is at http://www.bess.co.

Acknowledgments

Christopher Peri—We would like to thank all the unknown coders on the interwebs who have contributed to not only Twitter's success, but creating mountains of technical information and code examples that allows a lowly hobby programmer, like myself, to learn how to work with Twitter API and one day...write a book on it. A number of people have helped with this book, but I want to call out three people specifically: @chiah for creating the foundation of Hour 1, @jon_wu for Hour 8 as well as helping with debugging and general feedback on technical issues, and @LanceNanek for debugging and researching Android in Hour 22.

We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

You can email or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone or email address. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: opensource@samspublishing.com

Mail: Mark Taub
Associate Publisher
Sams Publishing
800 East 96th Street
Indanapolis, IN 46240 USA

Reader Services

Visit our website and register this book at www.informit.com/title/9780672331107 for convenient access to any updates, downloads, or errata that might be available for this book.

Preface

This book on the Twitter API is geared to the programmer who is just a bit past beginner—who knows the basics of LAMP, including how to set up a basic server, PHP, JavaScript, HTML, and CSS. You do not have to be an expert programmer to use this book, but you should know how to look things up. In writing this book, we have tried to provide you with everything you need to get a simple Twitter client up and running. We include an hour on setting up your environment, as well as providing you with HTML and CSS codes to have something up and running. However, it's beyond the scope of this book to explain what is happening with these codes. Instead, we focus on the code surrounding the API calls, OAuth, and the returns. That does not mean that you could not use this book if you are a beginner programmer. Because we provide you with all the code and build an application up step by step, you can stop at any time and look up parts of the code you do not understand. However, if you have never coded anything before, you may find that this book moves far too fast. It may be better to get an introductory book on basic programming in PHP before reading this book.

In writing this book, we also kept in mind experienced programmers who have been asked to create a Twitter application or include Twitter support in a current application, even if they do not know much about Twitter. We believe it's important to understand what Twitter is, how it's being used, and what makes it different from other social media services. It's with this understanding that you will be able to approach your Twitter project with a more engaged understanding of what your application is trying to accomplish, which is the best way to not only satisfy product requirements, but also design future growth.

Sams Teach Yourself Twitter API in 24 Hours is a little different from most technical books in that the book is geared around creating a functional Twitter client, including all HTML, CSS, JavaScript, and PHP needed to create your own application. We also dedicated the last four hours of this book to getting you started with making API calls on the iPhone and Android OSes in case you want to make your own mobile Twitter application.

Unlike most books, this book was written as Twitter and the API set was going through major changes. As such, the book and the code used in the book have been edited many, many times. So much so that we expect there will be a technical oversight here and there. So be sure to check the book's website for changes and updates

(http://www.twitterapi24.com/). Also, as much as we tried to keep up with all the changes happening with Twitter, we fully expect some details about the various API's to evolve from the time of the last edit to the time you have this book in your hands.

We hope you enjoy this book.

This page intentionally left blank

HOUR 3

Key Issues to Consider When Developing Twitter Applications

What You'll Learn in This Hour:

- Different types of Twitter users and how they impact code design
- Different types of Twitter applications and program architecture
- Things to consider if you are not building a web-based application

Types of Twitter Users

As one would expect with an API system as open as Twitter, and the explosion of interesting applications people have developed, we have also seen the development of different types of Twitter users. Understanding these types of users and knowing which of them we are trying to reach will inform how we may want to build our Twitter application framework. As with any large user base, there are a number of ways to set up categories. In this hour, we will break down and discuss the users in the following categories or types.

The News Reader

Twitter is a great source of breaking news, whether it's politics, business, sports, or following celebrities. Most users use searches to find what they are interested in, or they follow Twitter feeds that act like RSS readers. For example, BreakingNews is what you would guess it would be—a Twitter account publishing breaking news. Most news outlets have such accounts: CBSNews, ABC, BBC, and so on. The screenshot of NewsSnacker, an application created by the author (shown in Figure 3.1) is a good example of a Twitter application that focuses on the news.

FIGURE 3.1 Screenshot of NewsSnacker.

Although making search and Twitter account API reads from Twitter does not require authentication, you can still get dinged going over the API limit because Twitter will limit calls from an IP address. So, you still need to keep in mind how often you make calls. In the case of NewsSnacker, we use a white-listed account because the user could exceed the API calls-per-hour limit since each news service is a separate call. Suppose that the user has 10 sources and refreshes every 30 minutes. That is 200 calls in an hour, which is over the current limit of 150 for non logged in users. This does not include normal calls to check for new mentions or direct messages from the user's chosen Twitter client application. An alternative approach is to create a list of twitter news accounts and then call that list. However since newsSnacker removes duplicate posts, a large number of returns on the list call would be required. Both approaches have their merits however; one feature of newsSnacker is to allow a custom list of sources. This can be done by having the user log into the application and then select which of their lists they would like to call thus the second approach is being pursued in the next version of the application.

Chatters

Twitter does allow for people to have conversations; it's called a *direct message*. However, many people like to hold their conversations in public and a big attraction for these people is conversation threading. This is a very complicated proposition, so much so that new APIs are being created to deal with this situation. We will cover

retweeting in later hours, but this could cause quite an impact on your code's structure because of older reply techniques that use the letters RT for conversations instead of recent API methods that support replies formally. So, supporting Twitter conversation is a decision you will want to make early in your product's design.

Power Users and PR Managers

Although you will have a drag-out fight between the two because one is personal messaging and the other is more professional, the impact on product design is not that much different. PR (public relations) managers, power users, and anyone who consumes or monitors a lot of Twitter information will put special requirements on you as a product developer. Like the limits to the number of API calls mentioned in the section discussing the user group news readers, the issues with the power users and PR managers group will be the same, with the added requirements of being able to sort and search the stream of messages that come in. They may also need to send messages on a schedule or from people using the same account. There is usually no simple way around this issue other than to start thinking of a well laid-out database up front. You may want to also explore having your server make the API calls and relay the information to your Twitter application in the form of automatic processes or bots. Furthermore, set up your architecture to deal with a wide variety of API calls. We will cover this later in the hour. PR managers will want more than just searching the Twitter stream; they will want to make sense of it and make sense of who is on that stream and their influence. The API has just expanded to handle retweets, but not all Twitter clients will be updated to work with this API. As such, you still need to pay attention to RT (the current convention for a retweet) and hashtags. Plan for this up front. Also, plan to keep some of the user information in your database; you will want to use it for user profile and relationship analysis. Although the number of power users, compared to typical Twitter users, is quite low, having a power user using (and advocating) your application is highly desirable, and although every power user you talk to will have a different list of features and functions, there are some things you must be able to support—for example, dynamic search. Just providing a call and return to the search API is not good enough anymore. The current and future power users of Twitter are going to demand just as much power and feedback as they get using Google search. For example, power users would want links with the tweets that are returned to be followed and analyzed in some manner. Perhaps you should show a thumbnail of the site, or display the title and the first 50 words of the link. Be sensitive to nonstandard protocols, such as searching stock quotes using the $ sign in front of the stock market ID. For example, $aapl for Apple. Power users are going to demand speed and customization and will fully expect that your application understand the nonstandard features (social conventions) of Twitter.

Microbloggers

Microbloggers will want to take the time to craft each tweet carefully. Pay attention to the ease of creating a message—that is, allowing them to save as drafts, sending to multiple Twitter accounts, spell checking (yes, spell checking), and although this is not easy, a quick look up of the other tweeters or access to a list of tweeters. Especially for PR users, you may want to have a look at simple web-based CRM products to give you ideas. A new API to Twitter is the capability to store lists of tweeters. This is useful to all power users as well as microbloggers.

High-Frequency Users (TwitterHolics)

The current rules of the API system allow only 350 calls per hour if you are logged in, 150 if not. This may seem like a lot, but based on what features you are providing to your users, this can go very quickly. It's not unlikely that you could have five API calls per user action if you need to make follow-up calls. If they are high-frequency users, they may find themselves approaching the 350-call limit pretty quickly. Although there are calls that do not require credentials, you could still run up against this limit because Twitter does count the number of calls from an IP. As such, be sure you monitor the number of calls the user has left and deal with it accordingly. The good news is that an API call exists for checking how many API calls the user has left which does not count against your API limit. However, calling it over and over again too often (every 5 seconds, for example) could trigger other traffic limit controls.

New Users

This is less an API architecture question than a GUI issue. Although GUI design is not addressed directly in this book, consider using clear terms and common metaphors (like an email system, for example) for the layout and functionality of your application. Do not assume that your users will understand various social conventions in Twitter, so explain it up front and design your functions' intent clearly using tool tips for icons for example. If you are making an application that reflects some aspects of the Twitter.com site, be sure to follow the conventions Twitter uses.

Bots

Bots (programs that perform automated tasks), including creating spam or setting up phishing attacks, will always be an issue. A sophisticated Twitter application will be aware of some of these bots and try to protect users. You may, however, need to

create your own bots (for good, not evil). For example, you might take a RSS feed and republish it to Twitter after passing it through a business rules filter which is something the main Author of this book does. Because a bot is nothing more than "rules" you have for dealing with reading or creating Twitter messages or lists, you will find creating automated processes very easy with the Twitter API.

Types of Twitter Applications

Normally, when I'm about to start writing a Twitter application, I already know what I want it to do. Thus, based on the features and functions I have in mind, I already know what platform and category of users I'm targeting. Because we cannot know what you, the reader, have in mind, we will try to set up a basic framework for thinking about the various things you can do with Twitter as we go through this book. Part of Twitter's success is its simplicity and wide-open API. As such, people have developed powerful, sophisticated applications, mashups, and simple widgets that run in other apps or on web pages. However, the approach you will take building a full-on application is different from building a simple mashup or widget.

By the Way

A mashup is a web page or application that takes two or more data sources and combines them into a new service. Typically, mashups create a functionality not envisioned by the creators of the original sources. Twitter is a very popular mashup source.

Building a feature-rich Twitter application takes some planning. Although we will walk you through various examples of how to build apps around specific APIs, we want to bring focus, too. There is an overall approach you should determine before you write line one.

Widget

Let's talk about architecture around a simple widget. Suppose our simple widget is going to display the results of a search or the latest tweets from a user. This is the easiest to build. All we have to think about is four steps: make an API call to Twitter, parse the return, format it, and display it. That's it. We diagrammed this simple architecture in Figure 3.2.

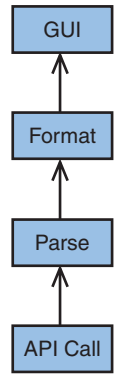

FIGURE 3.2
Example of a simple Twitter API diagram.

All API systems work this way, but what's great about Twitter is that the results are already of value. Quite often, blogging sites (mostly personal) have this type of widget. I have a widget like this on my blog (see Figure 3.3).

FIGURE 3.3
Screenshot of a Twitter widget on www.perivision.net/wordpress.

Mashup

Because a mashup can be the combination of anything, and that's kind of the point of mashups, we are going to think about our architecture a bit differently. Although technically, a mashup can be just two sources of information or very complex number and relationship of sources, we are going to stick with the spirit of what is considered a mashup by just thinking about mixing two data sources. For example, we can take our Twitter search feed and weather data and display tweets from places that are raining versus tweets from where it's sunny. In this case, we need to store our returns from Twitter somewhere while we get weather data. Then we need to perform some business logic on those returns.

Business logic is a nontechnical term generally used to describe the functional algorithms that handle information exchange between a database and a user interface. It is distinguished from input/output data validation and product logic. *From Wikipedia, the free encyclopedia*

By the Way

In this case, we need to hold our returns in an array so that when we get the weather data, we can reorganize our data. Because tweets are small, discrete messages, it makes sense to create a multidimensional array object that we can easily explore. So now, we will add one more layer to our diagram. As you can see in Figure 3.4, we are using arrays to store our parsed return so that we can apply some rules (business logic) to create a more valuable dataset.

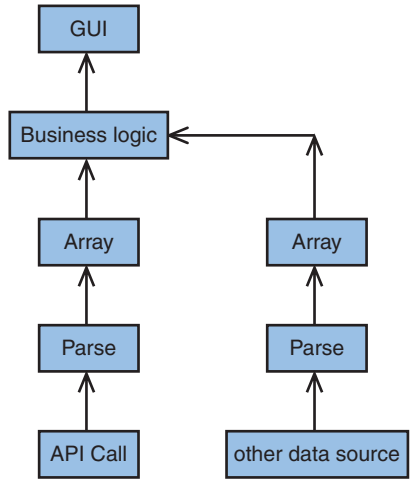

FIGURE 3.4 Example of combining two data streams.

Twitter Application

I would expect that only a small percentage of readers of this book are intending to build a full-featured Twitter client, but if you are, you want to approach building your application like any other application. Think about your calls to Twitter almost like calls to a database where you provide a set of parameters with your call and get a filtered response that can then be analyzed or applied to a set of rules. It is also well worth your time to set up your Twitter calls in a separate class to deal with errors and changes to the API. You should also set up another class to deal with converting your Twitter calls into multidimensional arrays and/or storing them in a database. The reason for this is that Twitter is still changing. Even during the writing of this book, we had to make adjustments to the book's index as new methods were introduced and other calls were deprecated. By keeping these two processes in standalone classes, you're going to save yourself some headaches down the road. If you are

planning on building a full-scale Twitter app, we recommend bookmarking the website for this book and the Twitter API site. Really! It changes and grows that much.

Also, somewhat like a database, you can store information in Twitter. For example, a much-overlooked feature is favorites. This API call allows you to save tweets. This can be quite useful as a means of understanding which tweeters and types of tweets a user tends to favor. New to the API list is lists. This is a list of tweeters a user creates. Again, it's a powerful bit of information that can be quite useful in understanding users' preferences. What is more interesting, though, is using these two API calls as storage devices if your user is under your control—a corporate account, for example. Because the user of that account does not interact with the account personally, you can use these API calls to store tweets and lists that can have greater meaning than originally intended. For example, suppose you have a corporate account for company X. We can store in the favorite list all tweets that match a certain rule, like any tweet that has an unfavorable term in the tweet. Now you have a list of tweets that public relations can examine using other than the application you developed. Also, remember you have access to user bio, location, and other elements. Again, because of how open Twitter is with its API, you can use these fields for anything—for example, including the updating of the Twitter background based on the latest message from the company, or perhaps updating the location field if you're a mobile food van, or changing the profile image based on the time of day or your mood. Instant database functionality ... of sorts! Now this does not mean you should not have a database if you intended on storing anything beyond the simple examples provided here. Also, it is not recommended to abuse this open access by placing unrelated data in these fields. Most applications will follow a simple structure, as illustrated in Figure 3.5.

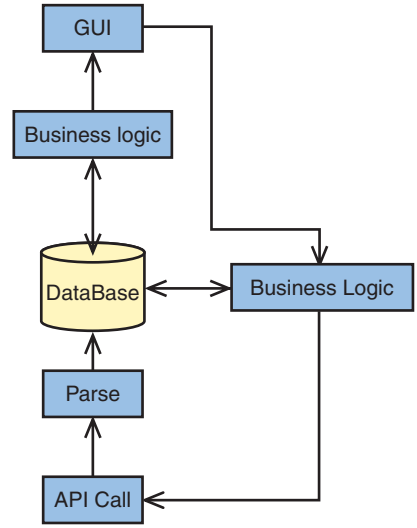

FIGURE 3.5 Example of an architecture placing a database between API calls and an application's business logic.

Pure Chat

This class of Twitter application is concerned with creating tweets, reading incoming tweets, searching Twitter, retweeting, setting/getting favorites, and displaying simple user account information. Everything can be done as a standalone command, meaning you do not need to store information outside of Twitter. Each command has only one or two API calls. The current Twitter.com main web page is this type of application. Since we do not need to keep track of a state or store data, we can create this application using nothing more than a simple collection of PHP calls. For this class of application, we want to think about our application as a series of standalone pages. It would be a good idea to use cookies on the user's computer in case you need to store last-seen dates or other simple pieces of information.

Structured Display

Very common with Twitter applications are the capabilities to save groups, perform more advanced searches, display only new information and some threaded conversations, and so on. Although some of these structured displays can be somewhat complex, the approach you would take as a programmer is not that much different. Many of these structured displays can be achieved without storing information on the server but by using API calls and cookies instead. Consider the following example: Suppose we want to display a column of unread tweets, tweets from our "top 10" friends, three or four saved searches, and your current favorites lists. All of these can be achieved by passing variables within the existing API calls. You will actually work far harder at the UI than the backend coding. For this class of application, we want to set up our code as a series of calls that are more or less self-contained. This will make dealing with the GUI less troublesome as redesign is requested or required.

Twitter Statistics

Collecting statistics from Twitter data provides great promise for research, improved discover and communications. However, this class of Twitter application is a bit harder. TwittFilter, another application created by this author is in this class as shown in Figure 3.6.

This class of Twitter application depends on creating new information through analyzing the return or returns from past Twitter calls, and storing or modifying this information on the server, typically in a database. This, however, is where we find Twitter to be the most interesting; because Twitter has such a large user base, you can gather enough data to infer information that does not have a direct user correlation. Did someone say mashup? For example, a very popular and now API-supported feature called "Trends" in Twitter is nothing more than a constant search

across all messages being sent to Twitter, displaying the terms with the highest rate of occurrence. However, because of the large user base and ease of creating tweets, Trends tends to be one of the first places that news breaks.

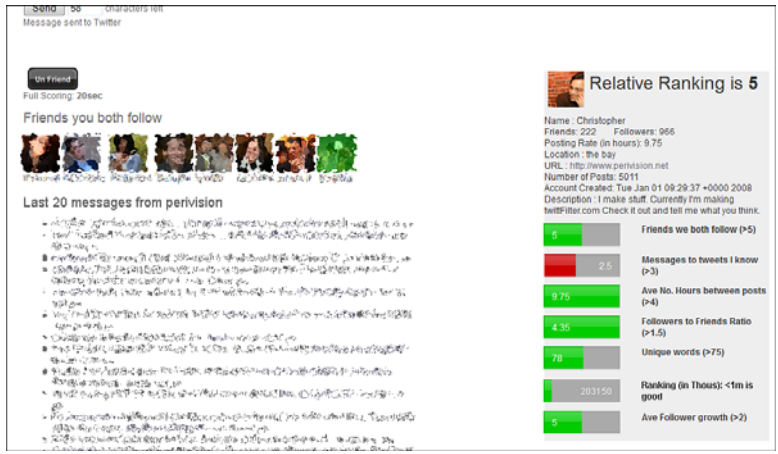

FIGURE 3.6 User Scoring screen of TwittFilter.

Because we need to store information as well as grab details for analysis, we need to think about how we structure our program differently. For this class of application, we want to think of Twitter as more of a database source. Setting up our arrays that allow for ease of use within formulas, as well as pulling and pushing into databases, will be a great benefit as our analytics become more and more complex. However, if you are not white listed, you will run into the API call limit quite quickly. It's recommended that if you plan to do statics that require large sample sets or recursive calls, that you explore the streaming API.

Platform

Now that we know the class of application we want to develop, we need to think about the delivery platform. If you are going to develop for UI-hosted apps, such as native mobile apps or Adobe Air, you may again want to modify how to approach your coding. Typically, when creating an app for the iPhone or other mobile platform, many of our UI elements are going to be handled on the device. You also will want to minimize the amount of traffic going back and forth as much as possible. Therefore, you should design your application around the output, which could be XML, JSON, or some custom bitcode. In this case, the organizational structure you choose will dictate how you structure your backend code. Because we have the luxury of storing information on our target platform, we can focus on speed and ease of architecture. Even though our backend code is not responsible for the presentation

layer (display), you still need to follow the basic tenants of good programming design by keeping the business logic separate from the API calls; don't fall into the trap of making each call from your application as a separate instance, as you may with the Pure Chat approach. You never know when your application starts to take on more features than you planned.

Summary

In this hour, you were introduced to various types of Twitter users. Depending on your product's target market, you may need to think about how you will approach the design architecture of your product.

This was not intended to be an exhaustive list, nor an absolute one. One could easily break this list into smaller pieces or roll it up into more general categories; instead, it's to provide a framework to think about the application you intend to build. We broke this up into two sections because we want to make a distinction between the type of use and type of application, However, do not think you can explore one without the other. When designing any application, you should always start with the user. What is the value proposition you are offering users in order for them to use your application? Once you understand that, you can then move to the type of application you want to create. So, we started out with an exploration of types of Twitter users, and then types of Twitter applications. We ended this hour with a short conversation about platforms. If you are developing for anything other than the desktop, you are most likely already aware of these points, but we included them for less-experienced developers as good to know.

Now, hold on to your hats because in the following hours, we are going to start building code!

Q&A

Q. ***Should I apply for a white-list account before I start coding?***

A. No. White listed accounts are currents not available. However, you will find that having 350 calls per hour is plenty as you learn how to develop your program.

Q. ***I plan to make a simple Twitter application now, but I may expand it later. Should I bother setting up a separate twitterAPI class?***

A. Yes. If you have any plans, even just thoughts of doing something beyond a few different types of API calls, set up a separate class for your API calls. In addition to new APIs, current API calls can change.

Workshop

Quiz

1. What is meant by thinking about Twitter as a type of database?
2. I check my Twitter account only a few days a week on my iPhone. What kind of Twitter user am I?
3. Is it illegal to create bots?
4. What is the easiest type of Twitter application to create?

Quiz Answers

1. This is a two-part answer: 1) Although Twitter exposes everything, you still can only get detail data on users one at a time although this is changing. Thus, thinking about accessing user statistics as if you were accessing a database is a useful way to think about what you can do with Twitter. 2) If you have control over the Twitter account(s), you can use the fields in Twitter to store information instead of on your database.
2. You are a news reader. Even if you are reading only your timeline (people you follow), you are more of a consumer of information than a creator.
3. No—and not all bots are bad. However, the good folks at Twitter do actively look for automated procesess that abuse the system.
4. A pure chat widget.

Exercises

1. Describe your typical target user and then determine the class of application you feel is appropriate for your user.
2. If you plan to create an automated process, write down each step and then count the number of times you will need to call Twitter to get information. What happens if the user hits refresh 10 times in 10 minutes? Will you go over the 150-API call limit if they are not logged in?

This page intentionally left blank

Index

Symbols

A

B

C

D

E

F

G

H

I

J-K

L

M

N

O

P

Q-R

S

T

U

V

W

X-Y-Z