Reference Guide for TKWatch+

Paul Donovan, Paul Kelly, and Michael Radovich
Loyola University Maryland
April 30, 2011

Abstract

This documents the program TKWatch+, submitted by us as an entry into a contest sponsored by TradeKing and Loyola University Maryland. It is not a user manual, nor is it a tutorial on the various application program interfaces (API's) and tools we used. Instead we document how to set up and run TKWatch+. This document was delivered along with all the Java source code, Javadocs for that code, and other resources required. These materials are available at [14]. We gratefully acknowledge the role of Dr. Paul Tallon in setting up the contest and the major programming and documentation assistance provided by Dr. George Wright.

Contents

1 Introduction **4**

2 Development **7**

2.1 Programming Environment 7

2.2 Application Program Interfaces 7

3 Operation **11**

3.1 Database Initialization 11

3.2 Starting `TKWatch+` 12

3.3 User Interface 12

3.4 Exercising `TKWatch+` 13

4 Issues **21**

4.1 Testing 21

4.2 Executable 21

4.3 Default Watch List Only 21

4.4 Database 21

4.5 XML 22

4.6 To-Do List 22

5 About Us **23**

5.1 Paul Donovan 23

5.2 Paul Kelly 24

5.3 Michael Radovich 25

1 Introduction

`TKWatch+` is a program for managing watch lists on TradeKing, an on-line brokerage firm for self-directed investors.[15] The program has three functions.

1. It stores watch list item particulars (stock market symbol, cost basis reference, quantity reference, and extended free text notation) in a relational database.
2. It supports adding, updating, and deleting watch list items on the user's TradeKing account.
3. At the user's option, it tweets additions and deletions to the user's Twitter account.

→ Code, Javadocs at [14]

This guide, along with accompanying source code and Javadocs, explains the set-up, development, and operation of `TKWatch+`.

The target audience for this document is a technically adept programmer. By that we mean an individual who:

- Has administrator privileges on the development platform;
- Can define environment variables for the host operating system;
- Can edit the `CLASSPATH` for the host operating system;
- Can operate in the `Eclipse`[5] integrated development environment (IDE) or something similar;
- Can manage user accounts on the host relational database; and
- Can operate at the command line prompt.

This guide is *not* a tutorial on any of the tools used during development. Instead we try to spell out the steps used to develop, set up, and run `TKWatch+` in the following sections.

Development : We discuss the hardware platform, software platform, development environment, and document preparation program we used. This includes configuration of all API's used.

Operation : We discuss initializing the database, navigating the user interface, and performing the basic functionality of adding, updating, and deleting a watch list item.

Issues : We enumerate all the remaining problems, limitations, areas for improvement, and the list of items that should be addressed in subsequent versions.

About Us : We close with a short biographical sketch about each of us.

→ Section 1, page 4

A major feature of this guide is cross-referencing. A cross-reference, such as the self-referential one to the left, appears as an arrow pointing to the referenced material. The cross-reference may refer to parts of this guide, to

Javadocs, or to source code.

References to documents, books, and web sites appear as numbers in square brackets. The numbered reference can be found in the References section on page 26. The index is on page 27.

2 Development

Legacy constraints determined the architecture of `TKWatch+`. We were limited to the technology provided by Loyola University Maryland and our faculty advisors. While we tried for portability, we failed to achieve it in some areas. We note those areas below.

2.1 Programming Environment

All development, testing, and operation was done exclusively on Loyola's Lenovo PC's running Windows Vista or Windows 7. We used two versions of `Eclipse`: Pulsar for Mobile Java Developers, build id: 20100218-1602, and Helios, Version: 3.6.1, Build id: M20100909-0800.[5]

2.2 Application Program Interfaces

We use nine API's in this project. We discuss them below in the order we began use them in the project.

2.2.1 Java

Since we programmed `TKWatch+` in Java, it could be taken for granted that we used the Java API.[11] To be specific, we used Java development kit `jdk1.6.0_23` with Java runtime environment `jre6` installed. The operating system environment must have the appropriate path and class path variables. These variables must refer to the appropriate Java executables and jar files.

2.2.2 Java Swing

`TKWatch+` is a client-based, stand-alone program which uses the Java Swing windowing graphical user interface (GUI).[21] In our code, we use Swing's "pluggable look-and-feel" capability to emulate Windows. This is, of course, a portability issue.

→ `Tkwatch.java`

2.2.3 JUnit

For unit testing, we used `JUnit`.[4] Specifically, we used the `JUnit3` API that comes packaged with `Eclipse`. It is also available as an open-source project.[6]. If you use another IDE, you will need to be sure that `junit.jar` is on the class path.

→ `WatchlistItemTest.java`

2.2.4 TradeKing

The TradeKing API, currently in beta, allows programmatic interaction with your TradeKing account.[17] Access to this API is available only to TradeKing account holders and must be requested from TradeKing. Upon request, you will receive a terms-of-use agreement. After signing and returning this agreement and upon approval by TradeKing, you will receive API credentials.

→ `tradeking.properties`

Rather than hard-code our credentials into the Java source code, we used a Java properties file to load the TradeKing credentials into a Java `Properties` object. The `tradeking.properties` file must appear in the directory holding the `TKWatch+` package `tkwatch`. It must contain the following entries.

`TRADEKING_ACCOUNT` : The eight-digit number of the user's TradeKing account. This is only necessary for making API calls that involve the account number.

`TRADEKING_APP_KEY` : The key for the `TKWatch+` application. This currently seems to common to all beta applications.[16]

`TRADEKING_URL` : The universal resource locator (URL) for accessing the TradeKing API. This too is supplied by TradeKing upon approval for API access.

`TRADEKING_USER_KEY` : This is the key associated with the user's TradeKing account login.

`TRADEKING_USER_SECRET` : This key is used to sign each request made against the TradeKing API.

Note that all the credentials in the accompanying file `tradeking.properties` are replaced with asterisks for security. You must replace the asterisks with your own credentials to run `TKWatch+`.

2.2.5 Codec

→ `Utilities.java`

The TradeKing API requires the `OAuth` protocol for signing each request. Our code handles signature via function `Utilities.generateSignature()`, adapted from [17]. This requires the Apache Commons Codec API for Base64 encoding/decoding.[2] The jar file `commons-codec-1.5.jar` must be on the class path.

2.2.6 Xerces

→ `Utilities.java`

The TradeKing API requests accept and return XML data. In some of our code we use routines adapted from [20] to convert XML data to and from Domain Object Model (DOM) documents. These routines require the Apache `Xerces` API.[1] The jar files `xercesImpl.jar` and `xml-apis.jar` must be on the class path.

2.2.7 Java Database Connectivity

`TKWatch+` stores enhanced watch list item data in a relational database, Microsoft's SQL Server in this implementation. Although this compromises portability, it is a legacy constraint. We used SQL Server 2005 Management Studio, version 9.00.3042.00.

We used Microsoft's version 1.2 JDBC driver for database connectivity through the standard JDBC API's available in the Java API.[9] The jar file `sqljdbc.jar` must be on the class path.

→ `database.properties`

Rather than hard-code our database credentials into the Java source code, we used a Java properties file to load the database credentials into a Java

Properties object. The database.properties file must appear in the directory holding the TKWatch+ package tkwatch. It must contain the following entries.

DATABASE_NAME : The URL of the JDBC data source for the watch list SQL Server database. In our implementation the URL is jdbc:sqlserver://localhost;port=1433;DatabaseName=watchlist. Address localhost can be replace with the Internet Protocol (IP) address of a remote server. Port 1433 is the default port for SQL Server. The default database name is watchlist. This default database name shouldn't be changed, because it's assumed when the database is configured.

→ Section 3.1, page 11

DATABASE_DRIVER : The JDBC driver for SQL Server connectivity. For Microsoft's version 1.2 JDBC driver, this is com.microsoft.sqlserver.jdbc.SQLServerDriver. This is a portability issue.

DATABASE_MASTER : The URL of the SQL Server master database, jdbc:sqlserver://localhost;port=1433;databaseName=master, assuming a default SQL Server installation. This is necessary when the database is configured. This is a portability issue.

→ Section 3.1, page 11

UID : The user ID for database access. For our implementation this is watchlist_user. This is necessary when the database is configured.

→ Section 3.1, page 11

PASSWORD : The password for database access. For our implementation this is g!st3rS1. This is necessary when the database is configured.

→ Section 3.1, page 11

2.2.8 Twitter4j

A feature of TKWatch+ is that it tweets watch list item adds and deletes to the user's Twitter account. Yusuke Yamamoto has created twitter4J, an unofficial library to support interaction of Java programs with the Twitter API.[23] The jar file twitter4j-core-2.2.1.jar must be on the class path.

2.2.9 Twitter

Like TradeKing, Twitter has released an API that exposes the Twitter service to programmatic access.[8, 18] Also as with TradeKing, Twitter requires credentials for authentication. Obtaining complete Twitter credentials takes several steps.

The first step is registering an application. This is done by clicking the "Register a new app" button on Twitter's "Twitter applications" page.[19] Successful registration furnishes the following credentials and URL's. Rather than hard-code our Twitter credentials into the Java source code, we used a Java properties file to load the Twitter credentials into a Java Properties object. The twitter4j.properties file must appear in the directory holding the TKWatch+ package tkwatch. It must contain the following entries.

→ twitter4j.properties

oauth.consumerKey : The Twitter registration process returns this both as the "API key" and as the "Consumer key." it needs to be entered into twitter4j.properties only once, as the consumer key.

`oauth.consumerSecret` : The Twitter registration process returns this as the "Consumer secret."

`TWITTER_REQUEST_TOKEN_URL` : The Twitter registration process returns this as the "Request token URL," `https://api.twitter.com/oauth/request_token`.

`TWITTER_AUTHORIZE_URL` : The Twitter registration process returns this as the "Authorize URL," `https://api.twitter.com/oauth/authorize`.

→ `GetAccessTokens.java`

Once the application is registered, the user still needs to obtain two more credentials. These can be obtained by running `GetAccessTokens.java`. This is a stand-alone program that can be run either at the command line or in `Eclipse`. It's easier to run in `Eclipse`, because it will be easier to cut-and-paste—which can't be done at the command line—than to type long keys. The following is the console output from a run of `GetAccessTokens.java`.

```
Open the following URL and grant access to your account:
http://api.twitter.com/oauth/authorize?oauth_token=
   ************************************
Enter the PIN(if available) or just hit enter.[PIN]:*******
Access tokens for *********
oauth.accessToken=************************************************
oauth.accessTokenSecret=****************************************
```

These last two items must be added to the `twitter4j.properties` file as follows.

`oauth.accessToken` : The access token used by the `OAuth` protocol to access the Twitter API.

`oauth.accessTokenSecret` : The access token secret used by the `OAuth` protocol to access the Twitter API.

Note that all the credentials both above and in the accompanying file `twitter4j.properties` are replaced with asterisks for security. You must replace the asterisks with your own credentials to run `TKWatch+`.

2.2.10 Document Preparation

In order to provide the extensive cross-listing, referencing, and indexing required for this document, we used LaTeX,the document preparation system developed by Leslie Lamport.[7, 13]. This document was prepared using the `refman` document class.[22] The execution itself was handled via PCTeXversion 6.1, a commercially available version of LaTeXmarketed by Personal TeX.[12].

3 Operation

3.1 Database Initialization

Before `TKWatch+` can be run, the database must be initialized. Before this can be done, a SQL Server administrator must create a login for user `watchlist_user`, password `g!st3rS1`. Then the administrator must set server roles for `watchlist_user` as shown in Figure 1. The watch list user

→ Section 2.2.7, page 8

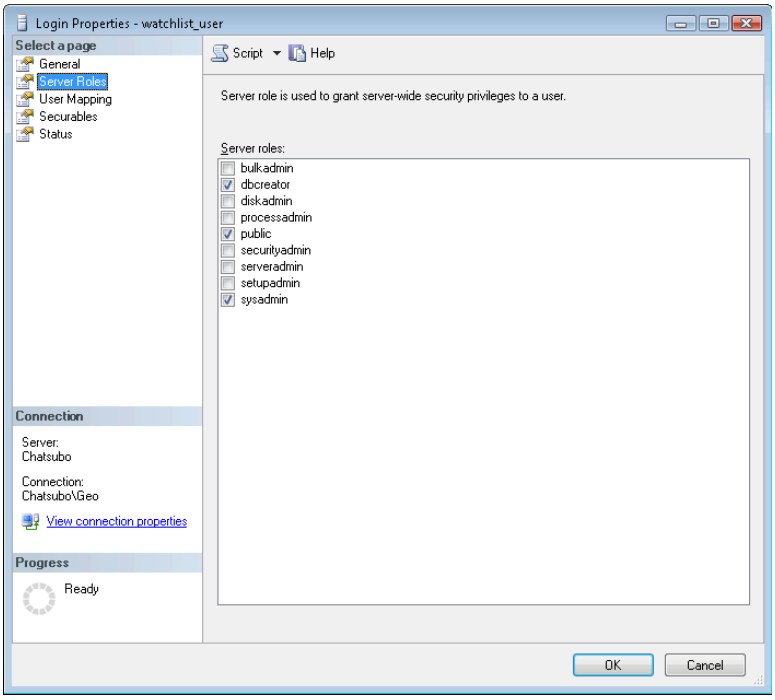

Figure 1: Server Roles for `watchlist_user`

account must be able to create and initialize the watch list database.

→ `Database.java`

Database initialization is done by running the `Database.main()`. This can be done either in `Eclipse` or at the command line. To run from the command line, change directories into the directory containing the `tkwatch` package, the three properties files, and the graphics files `poweredbyT4J.gif`, `tkwIcon.jpg`, `twitterColor.gif`, and `twitterGray.gif`. The command is `java tkwatch.Database`. The dialog box in Figure 2 will appear. You

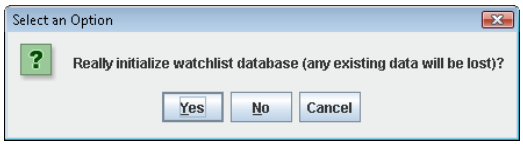

Figure 2: Database Initialization Option Dialog

should select "Yes." You should then see the dialog box in Figure 3. The database is now set up, empty, and ready to accept watch list items.

Figure 3: Database Initialized Message Dialog

3.2 Starting `TKWatch+`

As currently configured, `TKWatch+` can be run either within `Eclipse` or at the command line. To run from the command line, change directories into the directory containing the `tkwatch` package, the three properties files, and the graphics files `poweredbyT4J.gif`, `tkwIcon.jpg`, `twitterColor.gif`, and `twitterGray.gif`. The command is `java tkwatch.Tkwatch`.

3.3 User Interface

When `TKWatch+` starts, the user interface appears as in Figure 4. The user

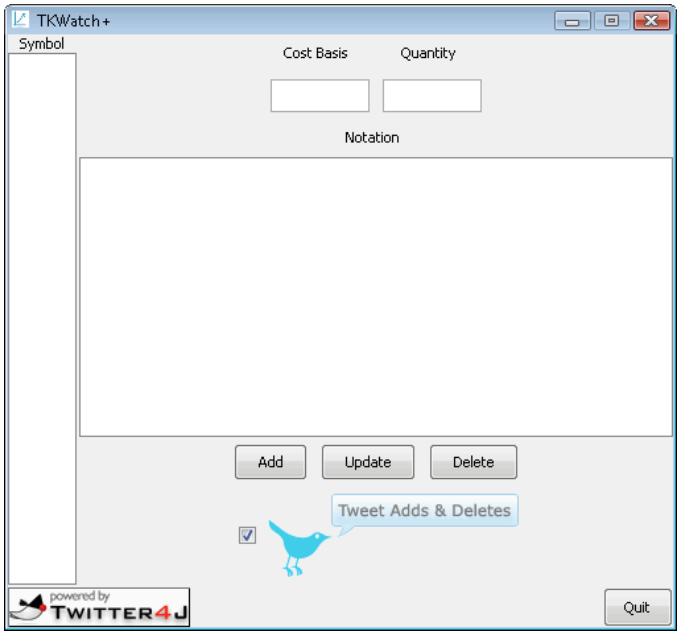

Figure 4: `TKWatch+` User Interface

→ `WatchlistPanel.java`

interface has eight features that either provide information to the user or allow the user to take some action.

Symbol List : The vertically-oriented area on the left, labeled "Symbol," is the list of market symbols for each instrument currently on the watch list. Since the database was just initialized, it is empty. When there is one or more symbol on the list, one of them is selected, initially the first. Data for that instrument appear in the next three interface elements.

Cost Basis : We are not sure what this data element is. According to [17] the cost basis element allows "... the user to associate values with the watchlist which assist in providing a portfolio value for the client." This element does not appear in TradeKing account displays of the watch list. It does appear in responses to the TradeKing API `/user/watchlists get` command.[17]

Quantity : We are not sure what this data element is. According to [17] the quantity element allows "... the user to associate values with the watchlist which assist in providing a portfolio value for the client." This element does not appear in TradeKing account displays of the watch list. It does appear in responses to the TradeKing API `/user/watchlists get` command.[17]

Notation : This field allows the user to create, read, update, and delete free text about the selected instrument. The current implementation limits it to 4096 characters.

→ `Database.java`

Add Button : This button allows the user to add an instrument to the watch list. Operation of this button is discussed below. If the "Tweet Adds & Deletes" check box is checked, the program tweets about the addition to the user's Twitter account.

→ Section 3.4, page 15

Update Button : This button allows the user to update an instrument in the watch list, i.e., change cost basis, quantity, notation, but not symbol. Operation of this button is discussed below.

→ Section 3.4.2, page 17

Delete Button : This button allows the user to delete an instrument to the watch list, i.e., change cost basis, quantity, notation, but not symbol. Operation of this button is discussed below. If the "Tweet Adds & Deletes" check box is checked, the program tweets about the deletion to the user's Twitter account. If checked, `TKWatch+` will tweet adds and deletions to the user's Twitter account.

→ Section 3.4.2, page 17

Tweet Check Box : This check box determines whether `TKWatch+` will tweet additions to and deletions from the watch list to the user's twitter account. The box is checked—i.e., tweeting is enabled—by default.

Quit Button : Clicking this button gracefully exits `TKWatch+`. Since some clean-up is performed by the action-handler for this button, this is the preferred method of exiting `TKWatch+`.

→ `Utilities.getQuitButton()`

3.4 Exercising `TKWatch+`

At this point, we assume that the user has successfully set up and started `TKWatch+` through Section 3.2. At this point, the user interface should appear as in Figure 4.

Before we exercise `TKWatch+`, consider the state of the database, the user's Twitter account, and the user's TradeKing watch list. See Figures 5, 6, and 7.

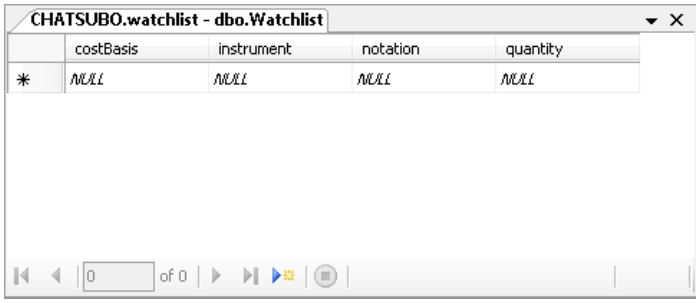

Figure 5: Empty Database

Figure 6: No Tweets From the User

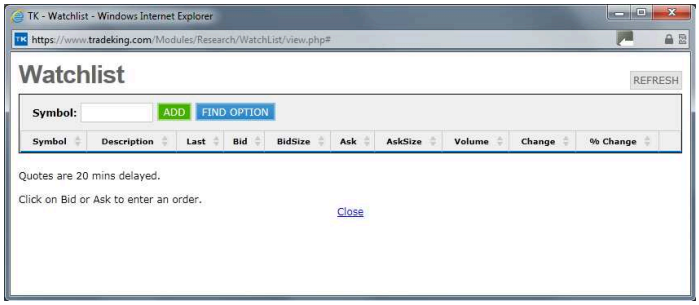

Figure 7: Empty Watch List

Note that there are no entries into the database yet, that there are no tweets from the user's timeline yet, and that the TradeKing watch list has no instruments watched. Now let's add, update, and delete some instruments.

3.4.1 Adding an Instrument

To add an instrument to the watch list, click the `Add` button shown in Figure 4. The interface now appears as in Figure 8. The purpose of the fields

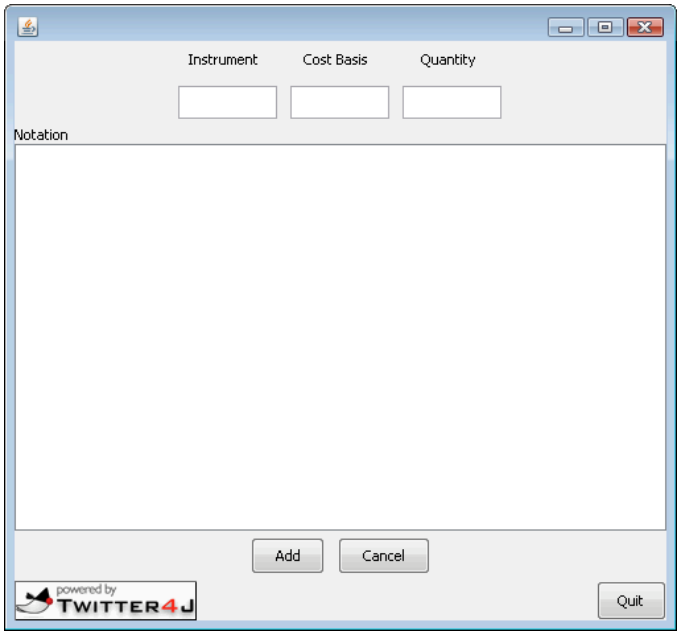

Figure 8: Interface for Adding an Instrument

in Figure 8 should be obvious. The user enters the instrument symbol, cost basis, quantity, and notation, then clicks the `Add` button. For demonstration purposes, we enter the data in Table 1.

Table 1: Test Data

Symbol	*Cost Basis*	*Quantity*	*Notation*
AAPL	111.11	100	First entry.
FSLR	222.22	200	Second entry.
GOOG	333.33	300	Third entry.
IBM	444.44	400	Fourth entry.

After entry of the data in Table 1, the database, the user's Twitter account, and the user's TradeKing watch list will look like Figures 9, 10, and 11.

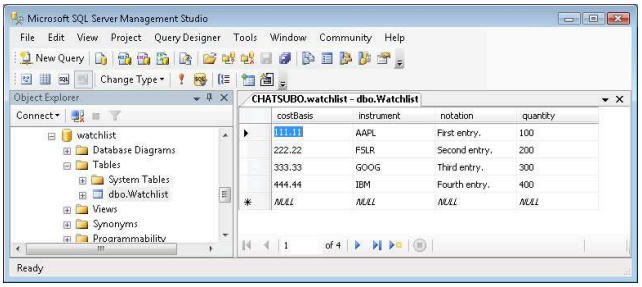

Figure 9: Populated Database

Figure 10: Four Tweets From the User

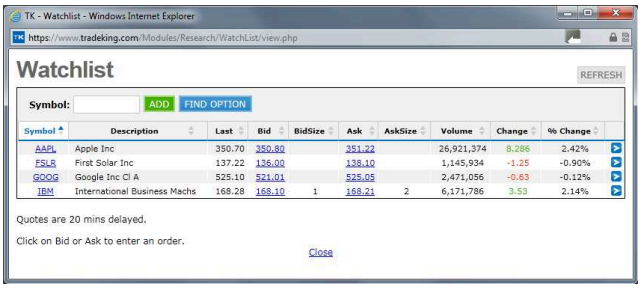

Figure 11: Populated Watch List

Note that there are now four entries into the database, that there are four tweets in the user's timeline, and the TradeKing watch list now features the four instruments from Table 1.

3.4.2 Updating an Instrument

To update an instrument in the watch list, first select the instrument to update by clicking its symbol in the `Symbol` window of Figure 4, for example, IBM. When you have selected it, make any changes you like in cost basis, quantity, or notation. For example, notice the edits in Figure 12. Then

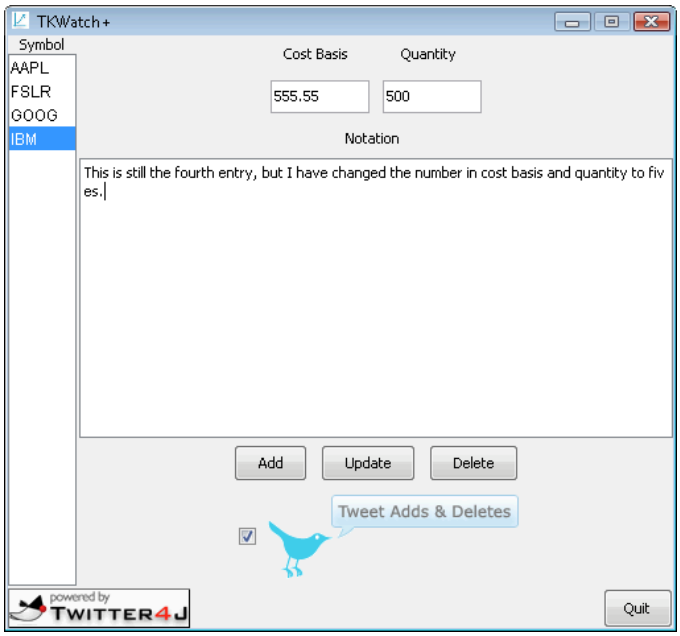

Figure 12: Updates Made to the IBM Entry

click the `Update` button. You will see this dialog. Click `Yes` to effect the

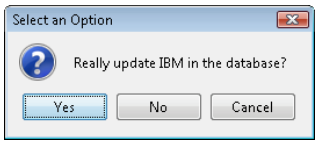

Figure 13: Database Update Option Dialog

→ Sectionconfirm, Page 22

changes. There is no confirmation dialog, but the database will now reflect the changes, as shown in Figure 14.

The update operation does not make any tweets. Moreover changes in cost basis and quantity are not apparent on the TradeKing watch list, because they are not displayed. Changes to cost basis and quantity *are* reflected in the TradeKing watch list as displayed in the return from the TradeKing `user/watchlists get` API command. (We omit the before and after displays of this command's results.)

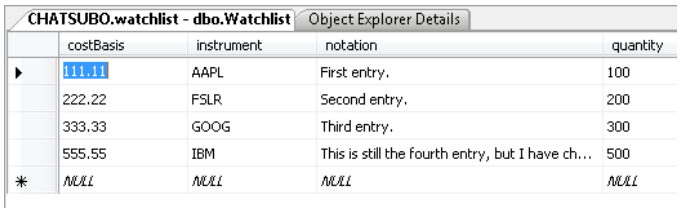

Figure 14: Database With Updated IBM Record

Note that you cannot edit the instrument symbol. To change the symbol, you must delete the old and add the new.

3.4.3 Deleting an Instrument

To delete an instrument from the watch list, first select the instrument to delete by clicking its symbol in the `Symbol` window of Figure 4, for example, IBM. When you have selected it, click the `Delete` button. You will see this dialog.

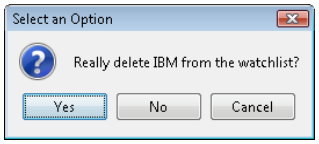

Figure 15: Database Delete Option Dialog

Click `Yes` to effect the delete. There is no confirmation dialog, but the database will now reflect the changes, as shown in Figure 16.

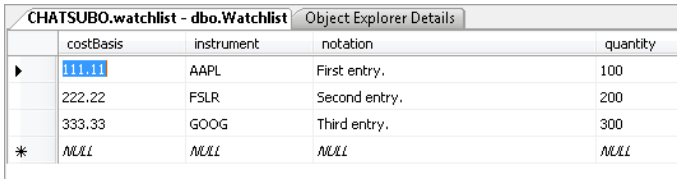

Figure 16: Database With IBM Record Deleted

The delete operation tweets the symbol deleted to the user's Twitter account, as shown in Figure 17.

Figure 17: Tweet About Deletion of IBM

The deleted instrument no longer appears in the user's TradeKing watch list, as shown in Figure 18.

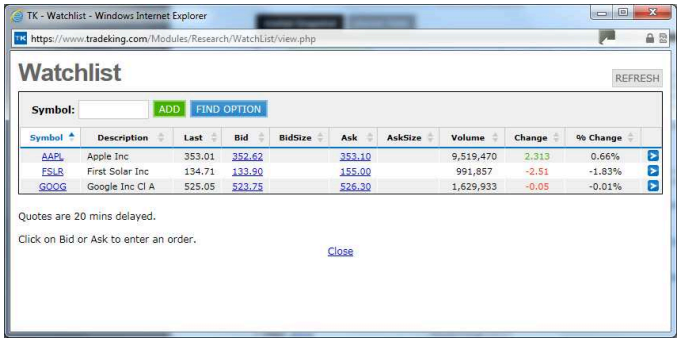

Figure 18: TradeKing Watch List No Longer Features IBM

4 Issues

TKWatch+was prepared in a very short time as an entry into a programming contest. We admit that this led us to cut corners in development. Here we discuss some of the program's shortcomings.

4.1 Testing

→ WatchlistItemTest.java

We did only a minimal amount of testing. We did unit testing on only one Java class. We did neither regression testing nor stress testing. The only operational or acceptance testing we did was in connection with producing the screen shots for this manual. The biggest resulting weakness is that we don't know how the program will respond to bogus inputs. What, for example, would happen if we tried to enter an non-existent symbol? The database would accept it, and we would tweet the add to the world, but we don't know what would happen in the TradeKing account.

We *did* uncover something interesting in the course of the testing we did though. Under certain circumstances, additions to the TradeKing watch list made via TKWatch+ could not be deleted from the watch list on the TradeKing account web site. They could only be deleted via TKWatch+. We did not have time to explore this or duplicate the behavior.

4.2 Executable

Currently TKWatch+ can be executed either in the Eclipse IDE or from the command line. Neither is a particularly elegant method. We tried packing TKWatch+'s Java classes and resources into a Java jar file. We kept running into class path issues. Neither we nor our faculty advisors could figure out how to solve these problems before the contest deadline.

4.3 Default Watch List Only

TKWatch+ currently handles only the TradeKing default watch list. Although it's not clear to us how one can create non-default watch lists from the TradeKing web site, [17] discusses named watch lists. The TradeKing API can be used to create and retrieve named watch lists. Enhancing TKWatch+ to handle multiple watch lists is possible, but it would require much more complex database processing.

Consider that one instrument could appear on many watch lists and one watch list could contain many instruments. This many-to-many relationship would have to be normalized with a junction record. Adds to and deletes from the database would have to take referential integrity into account.

4.4 Database

TKWatch+ currently is hard coded to work only with a SQL Server database installation. We thought it was necessary to store TKWatch+'s data in a database for several reasons.

First, it's the right way. It is *possible* to have TKWatch+ store and retrieve

watch list data from a flat, text file or perhaps a set of nested Java hash tables. But why? This is what databases are for. And besides, only database processing could efficiently handle named databases, as discussed above.

Second, we believe this is the way self-directed, technically proficient TradeKing account holders would want.

→ `Database.java`
→ `database.properties`

Third, portability is minimally compromised by having all database-specific code in `Database.java`. This file and, of course, `database.properties` are the only files that would have to be modified to handle another database.

Fourth, other databases are available. Our faculty advisor recommends that any further development of `TKWatch+` include alteration to use Apache Derby.[3]. Derby is a Java-based relational database that can be embedded in a Java application. Embedding the database in the application means that the user would not have to install, configure, or manage the database.

4.5 XML

→ `Utilities.java`

The TradeKing API accepts requests and returns results formatted as XML. Some of our code handles API results by parsing the XML text into DOM documents. Our code should also use XML processing to cast the body of the `user/watchlists update` API command. Right now we rely on less robust string-handling.

→ `Watchlist.java`

4.6 To-Do List

There are several loose ends in the code. Some have already been mentioned. All are marked in the source code with `TODO` comments. Here they are.

1. `Database.java`: SQL Server specific code is non-portable; consider embedding a Derby database.
2. `Tkwatch.java`: Windows OS look and feel is hard-coded; consider selecting OS-appropriate look and feel at run-time.
3. `Utilities.java`: Each panel has a message line, currently used only for the `Twitter4J` banner; consider displaying non-critical status information, such as confirmations.
4. `Watchlist.java`, `WatchlistItem.java`: Use XML handling where appropriate.
5. `WatchlistPanel.java` Add button currently opens a new window for add; use a panel in the main frame instead.
6. `WatchlistPanel.java`: Deleting all instruments leaves the last instrument's data in cost basis, quantity and notation fields; clean up after delete.

5 About Us

Paul Kelly, Paul Donovan and Michael Radovich met the first day of freshman year at Loyola University Maryland and have been roommates ever since. They quickly realized they shared a common interest in financial markets, technology, and entrepreneurship and have worked together on a number of projects while in school, both in the classroom and out. When not working on their next project they can often be found playing sports or arguing over sports teams they support. Working together on `TKWatch+` was both enjoyable and rewarding for them, and above all, a learning experience.

5.1 Paul Donovan

Figure 19: Paul Donovan, `pjdonovan@loyola.edu`

Paul J. Donovan is a current senior at Loyola University Maryland. He is a finance major within the Sellinger School of Business and Management and the Vice President of Loyolas Financial Management Association. For the past two summers he has interned at a private equity firm and an institutional money management firm, both in Boston. In his spare time he enjoys investing, reading, soccer, football, and the outdoors. When not at school he resides in Westwood, Massachusetts, with his parents, three siblings, and two black Labrador retrievers.

5.2 Paul Kelly

Figure 20: Paul Kelly, `pjkelly@loyola.edu`

Paul J. Kelly is a current senior at Loyola University Maryland. He is a marketing major and information systems minor within the Sellinger School of Business and Management. He currently works for Howard County Maryland, doing marketing research in an attempt to better the community and drive the local economy. In his spare time he enjoys cooking, baseball, board games, fantasy sports, and music. When not at school he resides in Westport, Connecticut, with his mother, three brothers, and his boxer, Molly.

5.3 Michael Radovich

Figure 21: Mike Radovich, `maradovich@loyola.edu`

Michael A. Radovich is a current senior at Loyola University Maryland. He is a finance major within the Sellinger School of Business and Management and a member of Loyolas Financial Management Association. For the past two summers he handled a variety of roles for a large beverage distributor in New York, working within their sales, marketing, and distribution departments. He currently writes investment articles for `MarketNewsVideo.Com` and `ETFChannel.Com`. In his spare time he enjoys traveling, soccer, hockey, and investing. When not at school he resides in Seaford, New York, with his parents, sister, and beloved Bichon Frise, Sophie.

References

[1] Apache. The Apache Xerces project. http://xerces.apache.org/. Online, accessed April, 2011.

[2] Apache Commons. commons ccodec. http://commons.apache.org/codec/. Online, accessed April, 2011.

[3] Apache DB Project. Apache Derby. http://db.apache.org/derby/. Online, accessed April, 2011.

[4] Kent Beck. junit. http://kentbeck.github.com/junit/. Online, accessed April, 2011.

[5] Eclipse Foundation. Explore the eclipse universe. http://www.eclipse.org/. Online, accessed April, 2011.

[6] JUnit.org. Resources for test driven development. http://junit.org/. Online, accessed April, 2011.

[7] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley Publishing Company, Reading MA, 1986.

[8] Kelvin Makice. *Twitter API: Up and Running*. Apress, Berkeley CA, 2006.

[9] Microsoft Download Center. Microsoft SQL Server 2005 JDBC Driver 1.2. http://www.microsoft.com/downloads/en/details.aspx?FamilyID=c47053eb-3b64-4794-950d-81e1ec91c1ba. Online, accessed April, 2011.

[10] Joel Murach and Andrea Steelman. *Murach's Java SE 6*. Mike Murach & Associates, Inc., Fresno CA, 2007.

[11] Oracle Technology Network. Java. http://www.oracle.com/technetwork/java/index.html. Online, accessed April, 2011.

[12] Personal TeX, Inc. PCTeXThe Integrated LaTeXEnvironment. http://www.pctex.com/. Online, accessed April, 2011.

[13] LaTeXProject. LaTeX—A Document Preparation System. http://www.latex-project.org/. Online, accessed April, 2011.

[14] Michael Radovich. Github repository. https://github.com/maradovich/TKWatch-Plus. Online, accessed April, 2011.

[15] TradeKing. Factsheets. http://www.tradeking.com/p/home/tradeking/about/facts.tmpl. Online, accessed April, 2011.

[16] TradeKing. Selective-Beta: Getting Started! http://community.tradeking.com/groups/tradeking-api/forum/topics/6415-selective-beta-getting-started/forum_posts. Online, accessed April, 2011.

[17] TradeKing. TradeKing API Reference Guide. Available as a PDF on request to TradeKing, after signing a terms-of-use agreement., March

2011.

[18] Twitter Developers. API Documentation. `https://dev.twitter.com/doc`. Online, accessed April, 2011.

[19] Twitter Developers. Twitter applications. `https://dev.twitter.com/apps`. Online, accessed April, 2011.

[20] Ajay Vohra and Deepak Vohra. *Pro XML Development with Java Technology*. O'Reilley Media, Inc., Sebastopol CA, 2009.

[21] Wikipedia. Swing (Java). `http://en.wikipedia.org/wiki/Swing_(Java)`. Online, accessed April, 2011.

[22] Graham Williams. The TEXcatalog online. `http://mirror.unl.edu/ctan/help/Catalogue/entries/refman.html`. Online, accessed April, 2011.

[23] Yusuke Yamamoto. Twitter4j. `http://twitter4j.org/en/index.html`. Online, accessed April, 2011.

Index

LaTeX, 10

Acceptance Testing, 21
Add an Instrument, 15
Add Button, 13
Apache Derby, 22
Audience, 4
Authors, 23

Button, Add, 13
Button, Delete, 13
Button, Quit, 13
Button, Update, 13

Check Box, Tweet, 13
Class Path, 7–9
Codec API, 8
Cost Basis, 12
Credentials, TradeKing, 7
Credentials, Twitter, 9, 10

Database, 21
Database Initialization, 11
Database User Server Roles, 11
Default Watch List Only, 21
Delete an Instrument, 18
Delete Button, 13
Derby, 22
Document Preparation, 10
Donovan, Paul, 23

Eclipse, 4, 7
Executable, 21

Issues, 21

Java API, 7
Java Database Connectivity (JDBC), 8
Java Swing, 7
JDBC, 8, 9
JUnit API, 7

Kelly, Paul, 24

Lamport, Leslie, 10

Notation, 13

Operation, 11

PCTeX, 10
Personal TeX, 10

Portability, 7–9, 21
Programming Environment, 7
Properties, Database, 8, 9
Properties, TradeKing, 8

Quantity, 13
Quit Button, 13

Radovich, Michael, 25
Regression Testing, 21

Starting `TKWatch+`, 12
Stress Testing, 21
Symbol List, 12

Terms of Use Agreement, 7
Testing, 21
Testing, Acceptance, 21
Testing, Regression, 21
Testing, Stress, 21
Testing, Unit, 21
To-Do List, 22
TradeKing, 4
TradeKing API, 7
Tweet Check Box, 13
Twitter API, 9
Twitter4j API, 9

Unit Testing, 21
Update an Instrument, 17
Update Button, 13
User Interface, 12

Xerces, 8
XML, 8, 22

Yamamoto, Yusuke, 9