

## Displaying Updated Stock Quotes

By **Douglas Lyon**

### Abstract

This paper describes how to extract stock quote data and display it with a dynamic update (using free, but delayed data streams). As a part of the architecture of the program, we make use of the observer, mediator and command design patterns. We also make use of the *JTable* and *JTableModel*. Finally we show how a multi-threaded update can be performed, using the web as a data source.

The methodology for converting the web data source into internal data structures is based on using HTML as input. This type of screen scraping has become a popular means of inputting data into programs. Simple string manipulation techniques are shown to be adequate for these types of well-formed data streams.

## 1 THE PROBLEM

Stock price data is generally available on the web (using a browser to format the HTML data). Given an HTML data source, we would like to find a way to create an underlying data structure that is type-safe and well formulated.

We are motivated to study these problems for a variety of reasons. Firstly, for the purpose of conducting empirical studies, entering the data into the computer by hand is both error-prone and tedious. We seek a means to obtain this data, using free data feeds, so that we can build dynamically-updated displays and perform data mining functions. Secondly, we find that easy to parse data enables us to teach our students the basic concepts of data mining using a simple first example. This example is used in a first course on network programming.

## 2 FINDING THE DATA

Finding the data, on-line, and free, is a necessary first step toward this type of data mining. Quotes have been available for years, from Yahoo. We can obtain the quotes, using comma-separated values (CSV) by constructing a URL and entering it into a browser. For example:

<http://finance.yahoo.com/download/javasoft.beans?SYMBOLS=AAPL,XLNX,ALTR,MOT,CY,CRUS,ADBE,MSFT,SUNW&format=sl>

This creates an output on the screen that looks like:

```
"AAPL",120.56,"6/28/2007","4:00pm",0.00,N/A,N/A,N/A,360814
"XLNX",26.95,"6/28/2007","4:00pm",0.00,N/A,N/A,N/A,0
"ALTR",22.16,"6/28/2007","4:00pm",0.00,N/A,N/A,N/A,0
"MOT",17.83,"6/28/2007","4:01pm",0.00,N/A,N/A,N/A,1000
"CY",23.36,"6/28/2007","4:05pm",0.00,N/A,N/A,N/A,0
"CRUS",8.32,"6/28/2007","4:00pm",0.00,N/A,N/A,N/A,0
"ADBE",40.42,"6/28/2007","4:00pm",0.00,N/A,N/A,N/A,0
"MSFT",29.83,"6/28/2007","4:00pm",0.00,N/A,N/A,N/A,804500
"SUNW",5.16,"6/28/2007","4:00pm",0.00,N/A,N/A,N/A,68305
```

The quote is typically returned in the form:

```
Symbol,price,date,time (20 min. Delay), change, open, bid,
ask, volume
"AAPL",55.31,"11/3/2004","4:00pm",+1.81,54.44,56.11,53.99,2150
4764
```

Before the market opens, these numbers return N/A. US markets are open from 9:30am-4:00pm, Eastern. If, in extended hours of trading (8am-8pm Eastern), there is trading (as shown for AAPL) the extended hours volume is listed. However, the price is as of the close of business + settlement time (which accounts for why CY was listed at 4:05pm).

To synthesize the URL needed to get the data, we use:

```
public static String makeQuoteURLString(String symbols[]) {
    String symbolsString = "";
    for (int i = 0; i < symbols.length; i++) {
        String symbol = symbols[i];
        symbolsString += ((i != 0) ? "," : "") +
symbol.toUpperCase();
    }
    final String qs =
"http://finance.yahoo.com/download/javasoft.beans?SYMBOLS=" +
symbolsString +
"&format=sl";
    System.out.println("qs:"+qs);
    return
        qs;
}
```

In order to fetch the data, given the URL, we write a simple helper utility that contains:

```
/**
 * Example:
 * <p/>
 * <code>
 * String s[] =
UrlUtils.getUrlString("http://www.docjava.com");
 * </code>
 *
 * @param urlString input urlString
 * @return array showing contents of web page.
 */
public static String[] getUrlString(String urlString) {
```



```
        Vector v = getUrlVector(urlString);
        String s[] = new String[v.size()];
        v.copyInto(s);
        return s;
    }
    public static Vector getUrlVector(String urlString) {
        Vector v = new Vector();
        BufferedReader br = null;
        try {
            URL url = new URL(urlString);
            br = new BufferedReader(new
                InputStreamReader(url.openStream()));
            String line;
            while (null != (line = br.readLine()))
                v.addElement(line);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (br != null) br.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        return v;
    }
}
```

The goal of such a program is to convert the URL into text, with one string per line, as retrieved from the web page. This is the core of the data retrieval phase.

### 3 ANALYSIS

In order to process the text data we need to decide how we are going to store and parse the data. To store the quote we create a *Quote* class:

```
public class Quote {
    private String symbol;
    private float last;
    private Date date;
    private float change;
    private float open;
    private float bid;
    private float ask;
    private long volume;...
```

With appropriate getters and setters. To store multiple quotes, we create a container class that has high-level quote processing methods:

```
public class Quotes {
    private Vector v = new Vector();

    public Quotes(String symbols[]) {
        this(Quote.getQuotes(
```

```

        UrlUtils.getUrlString(
            Quote.makeQuoteURLString(symbols)));
    }
    public Quotes(Quote[] e) {
        for (int i = 0; i < e.length; i++)
            add(e[i]);
    }

```

At this point, we build an ad-hoc parsing facility, based on a combination of string processing and a *StringTokenizer*:

```

/**
 * This constructor using a CSV based quote string, from
 * yahoo.
 * @param s
 */
public Quote(String s) {
    // assume that the quote is of the form
    // symbol price date time (20 min. Delay) change
open   bid   ask   volume
    //
"AAPL",55.31,"11/3/2004","4:00pm",+1.81,54.44,56.11,53.99,2150
4764
    StringTokenizer st = new StringTokenizer(s, ",");
    symbol = getSymbol(st);
    last = getPrice(st);
    String s3 = getSymbol(st);
    date = getDate(s3, getSymbol(st));
    change = getPrice(st);
    open = getPrice(st);
    bid = getPrice(st);
    ask = getPrice(st);
    volume = getLong(st);
}

```

The ad-hoc nature of the parsing scheme is even more evident when we examine the string manipulations needed to obtain low-level data types:

```

private static String getSymbol(StringTokenizer st) {
    String s = null;
    try {
        s = st.nextToken();
    } catch (Exception e) {
        return "";
    }
    return trimQuotes(s);
}

private float getPrice(StringTokenizer st) {
    final String s = getSymbol(st);
    if (s.indexOf("N/A") == -1)
        return Float.valueOf(s).floatValue();
    return Float.NaN;
}

private static long getLong(StringTokenizer st) {
    final String s = getSymbol(st);

```



```
        long l = 0;
        try {
            l = Long.valueOf(s).longValue();
        } catch (NumberFormatException e) {
            e.printStackTrace();
        }
        return l;
    }
}
```

The locally handled error handling can cause a general failure of robustness in the code. Of particular concern is what to do when the data format from the service provider changes. I suspect there will be a large cost associated with such a change, in terms of code rework.

## 4 DISPLAY

We are interested in a new “killer application” for development, called the *JAddressBook* program. This program is able to display stock quotes (and manage an address-book, dial the phone, print labels, do data-mining, etc.). The program can be run (as a web start application) from:

<http://show.docjava.com:8086/book/cgij/code/jnlp/addbk.JAddressBook.Main.jnlp>

And provides an interactive GUI for stock data.

symbol	last	date	time	change	open	day high	day low	vol
AAPL	122.04	6/29/07	4:00 PM	1.48	122.0	124.0	121.09	40485952
PCLN	68.74	6/29/07	4:00 PM	4.47	67.7	69.4	67.48	1748774
ABTL	4.25	6/29/07	4:00 PM	0.07	4.24	4.25	4.17	81201
SUNW	5.26	6/29/07	4:00 PM	0.1	5.2	5.33	5.16	104132288
NT	24.05	6/29/07	4:03 PM	-0.25	24.32	24.61	23.94	4195869
INTC	23.7399	6/29/07	4:00 PM	-0.1801	23.96	24.215	23.55	50607216
DVW	0.9	6/29/07	4:00 PM	0.0401	0.86	0.9	0.85	1129500
COMS	4.13	6/29/07	4:00 PM	-0.38	4.32	4.39	4.1	18967506
COH	47.39	6/29/07	4:01 PM	0.05	47.61	48.38	47.08	2873200
SONE	7.99	6/29/07	4:00 PM	-0.09	8.09	8.15	7.94	413385
DPZ	18.27	6/29/07	4:03 PM	-0.09	18.41	18.47	18.14	634300
SPOR	4.03	6/29/07	3:59 PM	-0.17	4.09	4.2	3.91	18913
SMG	42.94	6/29/07	4:03 PM	0.02	43.0	43.3	42.7	417400
ATLS	53.73	6/29/07	4:00 PM	-0.49	54.52	54.82	53.54	93658

Figure 4-1. The Stock Quote Viewer

Stock quotes are updated, dynamically, using a *JTable* and *JTableModel*, in Swing. Figure 4-1 shows an image of the *JTable*. In order to alter the list of stocks, the user selects the Edit:Customize menu. This displays a dialog box, shown in Figure 4-2. The list of stock symbols is stored in user preferences.

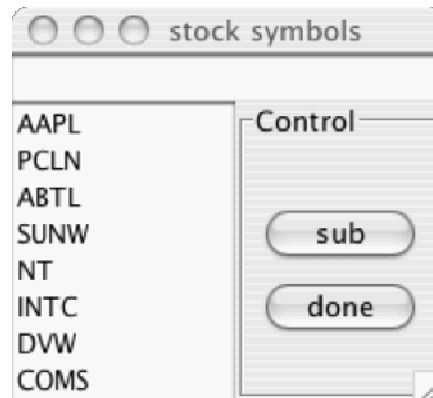


Figure 4-2. Stock Symbol Dialog

Figure 4-2 shows an image of the stock symbol dialog.

## 5 IMPLEMENTING A *JTABLEMODEL*

The *JTable* is updated, dynamically, by a threaded update to a *JTableModel*. The model is created by subclassing the *AbstractTableModel* as shown below:

```
public class StockTableModel extends AbstractTableModel {

    private String data[] [];
    private String[] columnNames = {
        "symbol",
        "last",
        "date",
        "time", "change", "open", "day high", "day low",
        "vol"};

    public StockTableModel() {
        updateDataArray();
    }

    public void updateDataArray() {
        Quotes quotes =
        Quotes.getQuotesFromTickerSymbolsBean();
        Quote q[] = quotes.getRecords();
        int n = q.length;
        data = new String[n][columnNames.length];
        for (int y = 0; y < n; y++) {
            Date date = q[y].getDate();
            data[y][0] = q[y].getSymbol();
            data[y][1] = q[y].getLast() + "";
            data[y][2] = DateUtils.getSimpleDateString(date);
            data[y][3] =
            DateUtils.getTimeStringHH_MM_PM_AM(date);
            data[y][4] = q[y].getChange() + "";
            data[y][5] = q[y].getOpen() + "";
            data[y][6] = q[y].getBid() + "";
        }
    }
}
```



```
        data[y][7] = q[y].getAsk() + "";
        data[y][8] = q[y].getVolume() + "";
    }
    super.fireTableDataChanged();
}

public int getRowCount() {
    return data.length;
}

public int getColumnCount() {
    return data[0].length;
}

public String getColumnName(int columnIndex) {
    return columnNames[columnIndex];
}

public Class<?> getColumnClass(int columnIndex) {
    return String.class;
}

public boolean isCellEditable(int rowIndex, int
columnIndex) {
    return false;
}

public Object getValueAt(int rowIndex, int columnIndex) {
    return data[rowIndex][columnIndex];
}

public void setValueAt(Object aValue, int rowIndex, int
columnIndex) {
    data[rowIndex][columnIndex] = (String) aValue;
    super.fireTableDataChanged();
}

public static void main(String[] args) {
    StockTableModel stm = new StockTableModel();
}
}
```

When the *fireTableDataChanged* method is invoked, observers of the *TableModel* use the *getValueAt* in order to update the cells in the table view.

## 6 JTABLE

The *StockTable* class uses a has-a relationship with the *StockTableModel* and the *JTable*. Since it connects the observer with the observable, we say that it has the role of the *mediator*;

```
public class StockTable {
    private StockTableModel tm = new StockTableModel();
    ClosableJFrame cf = new ClosableJFrame(new Date() + "");
}
```

```

    RunJob updateJob = null;
    private Stopwatch sw = new Stopwatch(1) {
        public void run() {
            cf.setTitle("Stock List (seconds since last
update: "
                        + getElapsedTime() / 1000 + ")");
        }
    };
    private Quotes quotes =
Quotes.getQuotesFromTickerSymbolsBean();

    public StockTable() {
        JTable jt = new JTable(tm);
        jt.setEnabled(true);
        jt.setAutoCreateColumnsFromModel(true);
        JScrollPane jsp = new JScrollPane(jt);
        cf.setSize(700, 300);
        cf.setContentPane(jsp);
        cf.setJMenuBar(getRunMenuBar());
        cf.setVisible(true);
        tm.addTableModelListener(new TableModelListener() {
            public void tableChanged(TableModelEvent e) {
                sw.reset();
            }
        });
        updateJob = new RunJob(1) {
            public void run() {
                if (cf.isVisible())
                    tm.updateDataArray();
            }
        };
    }

    private RunMenuBar getRunMenuBar() {
        RunMenuBar mb = new RunMenuBar();
        mb.add(getFileMenu());
        mb.add(getEditMenu());
        return mb;
    }

    private RunMenu getEditMenu() {
        RunMenu editMenu = new RunMenu("[Edit]");
        editMenu.add(new RunMenuItem("[customize{control c}"))
    {
        public void run() {
            displayTickerSymbolEditor();
        }
    });
        return editMenu;
    }

    protected RunMenu getFileMenu() {
        RunMenu fileMenu = new RunMenu("[File]");
        fileMenu.add(new RunMenuItem("[quit{control q}"))

```





```
        public void run(){
            updateJob.stop();
            cf.setVisible(false);
        }
    });
    return fileMenu;
}
```

The *command* design pattern is used for the menu items, as described in [Lyon 04B]. The *RunJob* uses the command design pattern applied to threads (known as *Project Imperion*) and is described in [Lyon 04C].

```
/**
 * This is called to update the tickers and the data in
the cells of the table.
 * It does not update the column labels.
 */
public static void main(String[] args) {
    System.out.println("starting stockTable");
    new StockTable();
    System.out.println("ending stockTable");
}

public void displayTickerSymbolEditor() {
    final TickerSymbolsBean ts =
TickerSymbolsBean.restore();
    ClosableJFrame cf = new ClosableJFrame("stock
symbols");
    cf.addComponent(new
RunListModelEditor(ts.getTickerList()) {
        public void run() {
            getTickerSymbols(ts,getValue());
        }
    });
    cf.pack();
    cf.setVisible(true);
}

private void getTickerSymbols(TickerSymbolsBean ts, Object
o[]) {
    ts.clearList();
    for (int i=0; i < o.length;i++)
        ts.addTicker((String)o[i]);
    ts.save();
    tm.updateDataArray();
}
```

## 7 RESOURCE BUNDLING

The stock symbols are stored in a serialized bean called the *TickerSymbolsBean*:

```
public final class TickerSymbolsBean implements Serializable {
```

```

protected int selectedTickerElement = -1;
//local vector for gui display
private Vector<String> tickerList = new Vector<String>();
private static final String prefs_key = "TickerSymbols";
private transient static PreferencesUtils prefs = new
PreferencesUtils(prefs_key);

private TickerSymbolsBean() {

protected void removeStockTicker(int index) {
    tickerList.removeElementAt(index);
    save();
}

//*****
*****
// addTicker uses checkTickerValid to determine if the
specified ticker
// is valid - displays a notification JOptionPane if not
//*****
*****
    public void addTicker(String ticker) {
        ticker = ticker.toUpperCase();
        if (checkTickerValid(ticker)) {
            tickerList.addElement(ticker);
            save();
        } else In.message("'" + ticker + "' is not a valid
ticker symbol");
    }

//*****
*****
// checkTickerValid will determine if a specified ticker is
valid
// by querying the finances.yahoo.com website, and searching
the
// html of the page for strings that will say the ticker is
not
// valid
//*****
*****
    private boolean checkTickerValid(String ticker) {
        String tickerCheckString =
buildTickerSearchString(ticker);
        Vector tickerPage =
UrlUtils.getUrlVector(tickerCheckString);
        String temp;
        for (int i = 0; i < tickerPage.size(); i++) {
            temp = (String) tickerPage.elementAt(i);
            if (temp.indexOf("is not a valid ticker symbol")
!= -1) {
                return false;
            }
        }
    }

```



```
        } else if (temp.indexOf("Error: Please enter a
ticker symbol.") != -1) {
            return false;
        }
    }
    return true;
}

private String buildTickerSearchString(String ticker) {
    return "http://finance.yahoo.com/q/op?s=" + ticker;
}

protected void printAllTickers() {
    PrintUtils.print(this.getTickerList());
}

//saveTickerList will save the vector tickernames into the
users preferences
public void save() {
    prefs.save(this);
}

public static TickerSymbolsBean restore() {
    Object o = prefs.restore();
    if (o == null) {
        System.out.println("Restore in TickerSymbolsBean
returns null:Warning!");
        return new TickerSymbolsBean();
    }
    return (TickerSymbolsBean) o;
}

public String [] getTickerList() {
    if (tickerList.size() == 0) {
        tickerList.addElement("aapl");
    }
    String s[] = new String[tickerList.size()];
    tickerList.copyInto(s);
    return s;
}

/**
 * clears the internal vector and zeros out the
 * element index.
 */
public void clearList() {
    tickerList.clear();
    selectedTickerElement = -1;
}

public static void main(String[] args) {
    TickerSymbolsBean ts = TickerSymbolsBean.restore();
    PrintUtils.print(ts.getTickerList());
    ts.addTicker("ibm");
    ts.save();
}
```

```
        ts = TickerSymbolsBean.restore();
        PrintUtils.print(ts.getTickerList());
    }
}
```

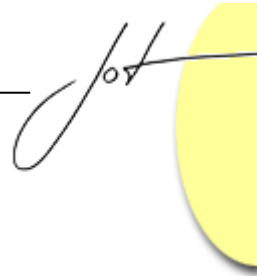
This makes use of the resource bundling techniques described in [Lyon 05A].

## 8 CONCLUSION

We show how an ad-hoc parsing technique can be used to obtain data from the web. The problem is that the way data is represented on the web is not consistent. Some sites are well-formed sources of data and others are verbose. For more sophisticated data mining tasks systems like NoDoSE are worth exploring [Adelberg].

The basic idea of obtaining data from the web and parsing it is a powerful concept. The web is a huge and growing source of data. However, we have found that relying upon others to keep data consistent, as a function of time, can create problems in a data-mining program. We have used Yahoo as a source of stock quotes and it has been reliable, both for the quality of service and its consistency of format. However, other sources of data have not been nearly as stable. The question of how to deal with ever-changing data formats remains open.

There are many sources of financial data on the web. For example, the Chicago Board Options Exchange (CBOE) now has a means to query option volume. Yahoo finance has historical end-of-day market data on individual stocks. These data sources can be very useful for the purpose of writing empirical finance papers or back testing trading strategies. The implementation of the data mining mechanism for these alternative sources of data remains a topic of future work.



---

## REFERENCES

- [Adelberg] Brad Adelberg, "NoDoSE—a tool for semi-automatically extracting structured and semistructured data from text documents", pp. 283-294, In SIGMOD 1998.
- [Lyon 04B] "Project Imperion: New Semantics, Facade and Command Design Patterns for Swing" by Douglas A. Lyon, Journal of Object Technology, vol. 3, no. 5, May-June 2004, pp. 51-64. [http://www.jot.fm/issues/issue\\_2004\\_05/column6](http://www.jot.fm/issues/issue_2004_05/column6)
- [Lyon 04C] "The Imperion Threading System" by Douglas A. Lyon, Journal of Object Technology. vol. 3, no. 7, July-August 2004, pp. 57-70. [http://www.jot.fm/issues/issue\\_2004\\_07/column5](http://www.jot.fm/issues/issue_2004_07/column5)
- [Lyon 05A] "Resource Bundling for Distributed Computing," by Douglas A. Lyon, Journal of Object Technology , vol. 4, no. 1, January-February 2005, pp. 45-58. [http://www.jot.fm/issues/issue\\_2005\\_01/column4](http://www.jot.fm/issues/issue_2005_01/column4)

## About the author



**Douglas A. Lyon** (M'89-SM'00) received the Ph.D., M.S. and B.S. degrees in computer and systems engineering from Rensselaer Polytechnic Institute (1991, 1985 and 1983). Dr. Lyon has worked at AT&T Bell Laboratories at Murray Hill, NJ and the Jet Propulsion Laboratory at the California Institute of Technology, Pasadena, CA. He is currently the Chairman of the Computer Engineering Department at Fairfield University, in Fairfield CT, a senior member of the IEEE and President of DocJava, Inc., a consulting firm in Connecticut. Dr. Lyon has authored or co-authored three books (Java, Digital Signal Processing, Image Processing in Java and Java for Programmers). He has authored over 30 journal publications. Email: [lyon@docjava.com](mailto:lyon@docjava.com). Web: <http://www.DocJava.com>.