

# **jDoji Candlestick Pattern Recognition Engine**

**Whitepaper**

Status: Draft

Date: August 10, 2010

## Table of Contents

INTRODUCTION.....	3
<i>What is jDoji engine</i> .....	3
<i>jDoji versions availability</i> .....	3
<i>jDoji versions comparison</i> .....	4
<i>Please send remarks and comments</i> .....	4
JDOJI API OVERVIEW .....	4
<i>jDoji class packages</i> .....	4
<i>jDoji classes</i> .....	4
JDOJI API DEVELOPER'S GUIDE .....	5
<i>Synopsis</i> .....	5
<i>Compatibility</i> .....	5
<i>Structure of the code and code snippets</i> .....	6
APPENDIX 1.....	7
<i>Example application code</i> .....	7

### Document revision history

<i>Date</i>	<i>Comment</i>
10-aug-10	Compatibility added; plus minor corrections.
8-Aug-10	First draft of the document. Relevant release: Beta.

## ***Introduction***

### **What is jDoji engine**

Sokyu Honma (1724-1803), was a rice merchant from Sakata, Japan who traded in the Ojima Rice market in Osaka during the Tokugawa Shogunate. In 1755 he wrote a book which outlined trading signals which are used up to nowadays to determine the changes in the market trend, allowing to benefit on short-term buy-sell contracts.

Sokyu Honma was trading rice futures, which was a new type of a deal at that time. He had a thorough approach to the data analysis. He not only included market volume and weather in his calculation, he also had a network of informers to have a geographic profile of the rice market. Today we have much better sources, and much better tools to calculate, however his legacy still exists in the method called “candlestick patterns” recognition.

It is often mistakenly said that the candlestick pattern recognition is a primitive form of technical analysis, which is very far from understanding the legacy of Sokyu Honma. His book written in 1755 was not about technical analysis, it was about the market psychology and how it develops. When speaking of yin and yang in the market prices (bullish, bearish) he finds that any yin has a seed of yang, whereas yang has always a seed of yin. That is, in plain speak, an expectation of increase when market is decreasing, and expectation of decrease when the market is increasing. This is close to life, but far from technical analysis. He also (very truly) noticed that it is the market trend change events that allow making big money, on the futures contracts in particular.

He formulated a number of patterns that can be used as market psychology indicators, mostly focusing on the trend reversal signals. These are the “candlestick patterns” as we know them today.

jDoji is a pre-configured pattern recognition engine which is packaged to recognize classical candlestick patterns. It works on any financial instrument where provided are historical prices for daily HIGH, LOW, OPEN and CLOSE. Aggregation period can be taken arbitrarily, “daily” is just the most common one. The engine is date agnostic, interval agnostic, instrument agnostic, and price scale agnostic. In order to make all that work, the pattern analysis is normalized using some necessary pre-processing, where we add into the picture instrument VOLATILITY20, SMA20, RSI14, and some other intermediate value vectors. Pattern recognition optimization itself, is implemented as a variant of a simple neural network, which allows very efficient calculation reduction for large sets of data, details of the calculation engine being fully proprietary.

### **jDoji versions availability**

“jDoji for Applet” is optimized for integration into a java applet. It is available as a standalone package ready for download. Other variants of the package can be created on request. For details please contact [jdoji@mouseclicktechnologies.com](mailto:jdoji@mouseclicktechnologies.com)

## jDoji versions comparison

Features/Versions	jDoji for Applet	jDoji for Servlet
High performance pattern factory	✓	✓
Auxiliary analytic indicators	✓	✓
Last value update mode	✓	✓
Thread safe storage		✓
Thread safe pattern factory		✓
Optimized for chart integration	✓	
Optimized for feed integration		✓

### Please send remarks and comments

Any additional comments and questions please send to [jdoji@mouseclicktechnologies.com](mailto:jdoji@mouseclicktechnologies.com).

## *jDoji API overview*

### jDoji class packages

jDoji consists of two packages: the storage package *com.mouseclicktechnologies.jdoji.storage* and the engine package *com.mouseclicktechnologies.jdoji*.

The storage package provides necessary classes implementing compound data buffer for historical time series, derived analyses, and intermediate calculation cache. The cache allows efficient constraint recognition optimization for large datasets and large pattern collections.

The engine package provides constraint factory and all public classes related to pattern recognition.

### jDoji classes

Full jDoji class reference is available in javadoc format at <http://www.jdoji.com/support/javadoc/>.

Most important classes are listed below.

#### *ICandlestickLoader*

This public interface is implemented by JDoji data loader which allows to provide timeseries data to the calculation engine. This interface must be used to load data into JDoji.

#### *IPattern*

This is public interface implemented by all candlestick patterns. This interface provides access to the patterns discovered as result of the data recognition. List of all predefined patterns is documented online at <http://www.jdoji.com/support/patterns/>.

#### *JDoji*

Main class. This class is not thread safe. This class must be instantiated to store raw data time series, and pre-processed data (additional data curves). This class will invoke the pattern factory for pattern recognition.*JDojiException*

Base class for all JDoji library exceptions.

*PatternCollection*

This class implements collection of predefined patterns.

*ResultElement*

This class links information about a candlestick and all the identified patterns.

*SearchResult*

Collection of ResultElements.

*TestReader*

Example parser used only for test case data

*TextReader*

Example parser used in the example jDojiApplet.

## ***jDoji API developer's guide***

### **Synopsis**

“jDoji for Applet” is a pattern recognition library. Patterns included in the library are documented at <http://www.jdoji.com/support/patterns/>.

“jDoji for Applet” is not thread safe, and it requires one object instance for each candlestick data series. Data series must contain sufficient history before the recognition can start. Pre-defined history depth necessary to determine previous trend is 20 data points (can be days, or any other time intervals). After 20 data points the recognition starts and scans to the end of the data series, producing the set of recognized patterns.

Data in the storage buffer must be sorted on date, otherwise the recognition will be executed on the sequence “as is”. If data upload is not sorted on data, it can be sorted after upload using auxiliary *sortData()* method. If data load is done in ascending order, then *sortData()* is not necessary.

### **Compatibility**

jDoji library is written and compiled based on JDK 5 standard.

Please report any compatibility problems to [jdoji@mouseclicktechnologies.com](mailto:jdoji@mouseclicktechnologies.com).

## Structure of the code and code snippets

Generic structure of the code should be as follows:

- Load raw data into the data buffer
- Execute the recognition on the data buffer (search patterns)
- Examine the result data collection

Here is example code snippet which does the job:

```
DateFormat format = new SimpleDateFormat("dd. MM. yyyy");
JDoji api = new JDoji();
ICandlestickLoader loader = api.getLoader();
TextReader reader = new TextReader(loader);
reader.parseFile("input_data.txt");
SearchResult result = api.searchPatterns();
for (ResultElement element : result) {
    System.out.println("Found patterns at " +
format.format(element.getCandlestick().getDate()));
    for (IPattern pattern : element) {
        System.out.println("Id: " + pattern.getId() + ", Name: " +
pattern.getName());
    }
    System.out.println();
}
```

Data loader implementation (*TextReader()* in this case) should upload the data points one by one, as follows:

```
loader.setDate(dayDate);
loader.setOpen(dayOpen);
loader.setHigh(dayHigh);
loader.setLow(dayLow);
loader.setClose(dayClose);
loader.commitCandle();
```

Result of the recognition will contain necessary identity values allowing to position the recognition result within the data set loaded into the buffer. Either sequence number or the date can be used to match the result to the original data series. The candlestick pattern will be identified by the four digit numerical identity, and the pattern name. All the patterns are documented at <http://www.jdoji.com/support/patterns/>.

## APPENDIX 1

### Example application code

```
import com.mouseclicktechnologies.jdoji.ICandlestickLoader;
import com.mouseclicktechnologies.jdoji.IPattern;
import com.mouseclicktechnologies.jdoji.JDoji;
import com.mouseclicktechnologies.jdoji.ResultElement;
import com.mouseclicktechnologies.jdoji.SearchResult;
import com.mouseclicktechnologies.jdoji.TextReader;
import java.io.File;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Main {

    public static void main(String[] args) {
        try {
            // Date format for output.
            DateFormat format = new SimpleDateFormat("dd.MM.yyyy");

            // Initialize jDoji object.
            JDoji api = new JDoji();

            // Obtain loader for data.
            ICandlestickLoader loader = api.getLoader();

            // Create reader and load data to jDoji.
            TextReader reader = new TextReader(loader);
            if (args.length == 1) {
                reader.parseFile(args[0]);
            } else {
                System.out.println("File argument not found");
                return;
            }

            // Sort data (only if required).
            api.sortData();

            // Search predefined patterns.
            SearchResult result = api.searchPatterns();

            // Output found patterns to console.
            for (ResultElement element : result) {
                System.out.println("Found patterns at " +
format.format(element.getCandlestick().getDate()));
                for (IPattern pattern : element) {
                    System.out.println("Id: " + pattern.getId() + ", Name: " +
pattern.getName());
                }
                System.out.println();
            }
        } catch (Exception ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

This code is included in the package as “TestJDoji.java” together with a sample data set “data\_example.txt” to produce a working demo application. Once compiled, it can be executed in a command line as “**java -jar TestJDoji.jar data\_example.txt**”.