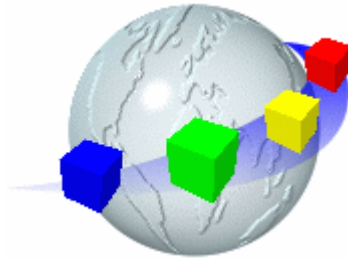


WebCab Technical Analysis

WebCab
Components



Preface

This documentation accompanies the WebCab Technical Analysis J2SE Component. WebCab Technical Analysis is a growing collection of functionality which enables the development and testing of technical trading systems. In particular, we will cover technical indicators, standing trading systems, performance measures and style analysis.

The first chapter of this documentation contains a brief introduction to the most important implemented features and related system requirements. In chapter two we let the developer quickly get started with deploying the component. We also detail how the functionality of this component can be tested through connecting to online web service demos hosted at WebCabComponents.com. The third chapter contains the mathematical documentation, which represents the theoretical background of this component's implemented features. Chapter four contains the programmer's guide containing a road map for developers to take advantage of every feature and capability. In chapter five, we describe the QA Clients which demonstrate the application of each of the methods within each of the classes. Finally, we introduce WebCab Components, its philosophy and approach to serving the JavaTM development community with robust and powerful J2EE applications.

If you have any suggestions for questions or queries concerning the use of this components then please feel free to contact us at:

<http://www.webcabcomponents.com/support/index.php>

Good luck with your project and thank you for your interest in our components.

The WebCab Components Team

Contents

Preface	i
1 Introduction	1
1.1 Product Description	1
1.1.1 Overview	1
1.1.2 Details	1
1.2 Package Details	2
1.3 Prerequisites and Compatibility	3
1.3.1 System Requirements	3
1.3.2 Compatibility	3
2 Where do I Start?	4
2.1 Using the J2SE JAR file	4
2.1.1 Compilation	4
2.1.2 Packaging	5
2.2 Testing the Component	8
2.2.1 Accessing an Online Demo	8
2.2.2 Compatible Web Containers	8
2.2.3 Selecting a method to test	8
2.2.4 Inputing data	8
2.3 Installing the Web Application Example Locally	9
3 Technical Trading Indicators	11
3.1 Moving Averages	11
3.1.1 Simple, Median and Geometric Moving Averages	12
3.1.2 Weighted Moving Averages	13
3.1.3 Variable Moving Average (VMA)	14
3.1.4 Kairi Indicator	14
3.1.5 Moving Average Based Trading systems	15
3.2 Directional Movement Indicator (DMI) and Average Direction Indicator (ADX)	16
3.2.1 Comparing Ranges	16
3.2.2 Positive Directional Movement (PDM) and Minus Directional Movement (MDM)	17

3.2.3	True Range of the Current bar (TR)	18
3.2.4	Defining the Directional Movement Indicators	18
3.2.5	DMI Trading System	18
3.2.6	Directional Indicator (DX) and Average Directional Indicator (ADX)	19
3.3	Accumulation/Distribution Indicators	19
3.3.1	Accumulation/Distribution Indicator	19
3.3.2	Chaikin Oscillator	20
3.3.3	Chaikin Money Flow (CMF)	21
3.4	Trend or Range?	21
3.4.1	Aroon Indicator	22
3.5	Market Strength	22
3.5.1	Balance of Power Indicator (BOP)	22
3.6	Oscillators	23
3.6.1	Money Flow Index (MFI)	23
3.6.2	Momentum and Rate of Change (ROC) Indicators	24
3.7	Bollinger Bands	24
3.8	Mean Reversion	25
3.8.1	Commodity Channel Index (CCI)	25
3.9	Stochastics	26
3.9.1	Interpretation and Application	26
3.9.2	Evaluation	26
3.10	Filters	27
4	Programmer's Guide	28
4.1	Developing with J2SE	28
4.1.1	Stand-alone Java Applications	28
4.1.2	Java Applets	30
4.2	J2EE™ Solutions Using J2SE Components	31
4.2.1	Writing JSP Pages	31
4.2.2	Designing EJB™ Components with J2SE Components	32
4.3	Support for Developers	34
4.3.1	Compilation and Custom Modification of Source Code	34
4.3.2	Online	34
5	Examples	35
5.1	Question and Answer (QA) Client Examples	35
5.1.1	Overview	35
5.1.2	Structure of QA Examples Directory	35
5.1.3	Top Four levels of the QA Directory Structure	36
5.1.4	Bottom Two levels of the QA Directory Structure	36
5.1.5	Quick Start Guide	36
5.1.6	Explanation of the QA Directory Structure and its files	37
5.1.7	Remarks on Java compilers	38
5.2	Custom Examples	39

5.2.1	Database Example with JDBC Mediator	39
6	Guide to WebCab Components	41
6.1	The Company	41
6.2	Presentation of Products	41
6.3	Supported Clients, IDEs, Containers and DBMSs	41
6.4	Transparent Functionality	42
6.5	Code Conventions	42
6.6	Company Culture and Activity	42
6.7	Product Life cycle	42
6.8	Support, Warranty and Upgrades	42

Chapter 1

Introduction

1.1 Product Description

1.1.1 Overview

Provides a collection of technical indicators which can be used in the construction of technical trading systems. Moreover, by using these methods with our JDBC mediator you will be able to iteratively apply these indicators to historical data stored within a DBMS.

1.1.2 Details

Within this J2SE Component we have implemented the following functionality:

- **Technical Indicators**
 - **Moving Averages** - Simple, Median, Geometric Moving Averages, Linearly Weighted Moving Average (LWMA), Exponentially Weighted Moving Average (EWMA), Variable Moving Average (VMA)
 - **Directional Movement Indicator (DMI) and Average Directional Movement Indicator (ADX)** - Directional Movement (PDM, MDM), True Range (TR), DMI Trading System, Directional Indicators (DX, ADX)
 - **Accumulation/Distribution** - Accumulation/Distribution Indicator, Chaikin Oscillator, Chaikin Money Flow (CMF)
 - **Trend or Range?** - Aroon Up/Down, Aroon Oscillator
 - **Market Strength** - Balance of Power (BOP)
 - **Oscillators** - Money Flow Index (MFI), Momentum, Rate of Change (ROC)
 - **Bollinger Bands** - Upper and Lower Bollinger Bands
 - **Mean Reversion** - Commodity Channel Index (CCI)
 - **Stochastics** - (fast and slow) %K Stochastic, (general) %D Stochastic
- **Filters** - Typical and Median price

This product also contains the following features:

- **GUI Bundle** - we bundle a suite of graphical user interface JavaBean components allowing the developer to plug-in a wide range of GUI functionality (including charts/graphs) into their client applications.
- **JDBC Mediator** - A J2SE Component which mediates between a J2SE component, its J2SE Clients and the Database server. The JDBC Mediator J2SE classes are a convenient way of enhancing all financial and mathematical specific methods with JDBC-based functionality. Check the `jdbc` subpackage of every J2SE class for JavaDocs documentation.
- **Web Application Example** - A Java WAR file which contains a JSP example that makes use of the functionality provided by our J2SE Component.
- **Synthetic JDBC** - The JDBC functionality provided by the Web Application example included within this package. This Web Application is an example of how to make a JSP client using our J2SE Component while manually implementing the JDBC code. The JSP Application applies J2SE methods to certain rows from the database and lists the output in HTML format.

1.2 Package Details

The WebCab Technical Analysis J2SETM Component package has the following structure:

- Introductory Text File (README.TXT)
- License Agreement
- Documentation in PDF Format
 - Product Description
 - System Requirements
 - Compatibility Issues
 - Deployment Guide (How to get started?)
 - Mathematical Documentation
 - Programmer's Guide
 - Examples
 - Guide to WebCab Components
- JavaDocs Documentation
 - Class Descriptions
 - Methods Descriptions

- Deployment Archives (Java JAR file)
- Examples and Related Source Code Files
- WebCab Components J2SE Brochure

1.3 Prerequisites and Compatibility

1.3.1 System Requirements

- An Operating System running Java™
- Pentium® III 500 MHz
- 128MB RAM

1.3.2 Compatibility

Operating System for Deployment

- Windows 2003, XP, 2000, NT, 9x
- Sun Solaris
- Linux/Unix
- IBM AIX
- HP-UX

Component Type

- JavaBeans™/Java API Library

Built Using

- Java™ 2 SDK Standard Edition 1.3.1/1.4.x

Compatible IDE Tools

- Borland JBuilder
- BEA Studio
- Sun One IDE (formally Forte for Java)
- Eclipse
- Oracle JDeveloper

Chapter 2

Where do I Start?

Start using the Technical Analysis J2SE Component right away by following the few quick and simple steps described in this chapter. If you require additional information regarding the use of this J2SE Component, then please don't hesitate to contact us via our support forum at: <http://www.webcabcomponents.com/support/index.php>

2.1 Using the J2SE JAR file

The Technical Analysis component comes wrapped inside a Java JAR file located inside the */Deployment* directory of this package¹. This class library contains all classes that provide the functionality offered by this J2SE Component as documented inside the API reference directory in this package (*/JavaDocs*). Regardless of the Java solution you are developing you will need to include this file within your project, or distribute it with your completed Application. Please note that whenever using the `J2SE-JARFile.jar` name throughout the next pages we will be referring to this Java JAR file located inside the */Deployment* directory.

2.1.1 Compilation

When compiling a Java solution that makes use of this J2SE Component you will need to make sure that the path to the J2SE JAR file is specified inside the `CLASSPATH` environment variable. The two most popular ways to compile an application are through calling the compiler via the command line or by using an IDE via JBuilder.

Compiling at the Command Line

If you compile and run your Java classes at the command prompt you should make sure that the `CLASSPATH` environment contains the path to the J2SE JAR file. In order to accomplish this under Windows you will need to type the following line at the command prompt:

¹If you are using the Demo version of this product, the name of the J2SE JAR file is `TADemoJ2SE.jar`. The name of the J2SE JAR file for one or more developers and Site-wide licensed products is `TAJ2SE.jar`.

```
set CLASSPATH=%CLASSPATH%; J2SEPath
```

where *J2SEPath* is the full path to the J2SE JAR file such as:

```
C:\My Downloads\TA\Deployment\TAJ2SE.jar
```

Under Linux the corresponding line is:

```
export CLASSPATH=$CLASSPATH: J2SEPath
```

Once you have correctly set the **CLASSPATH** you may invoke the **javac/java** tools to compile and run your Java classes. Another way to set the **CLASSPATH** variable is by directly passing its contents to these tools when typing them at the command prompt. For example, under Windows you would type:

```
javac -classpath %CLASSPATH%; J2SEPath ClassName.java
```

which would compile the *ClassName.java* file with the specified classpath, without affecting the global classpath after the compilation is done. The equivalent for the Linux **javac** tool is:

```
javac -classpath $CLASSPATH: J2SEPath ClassName.java
```

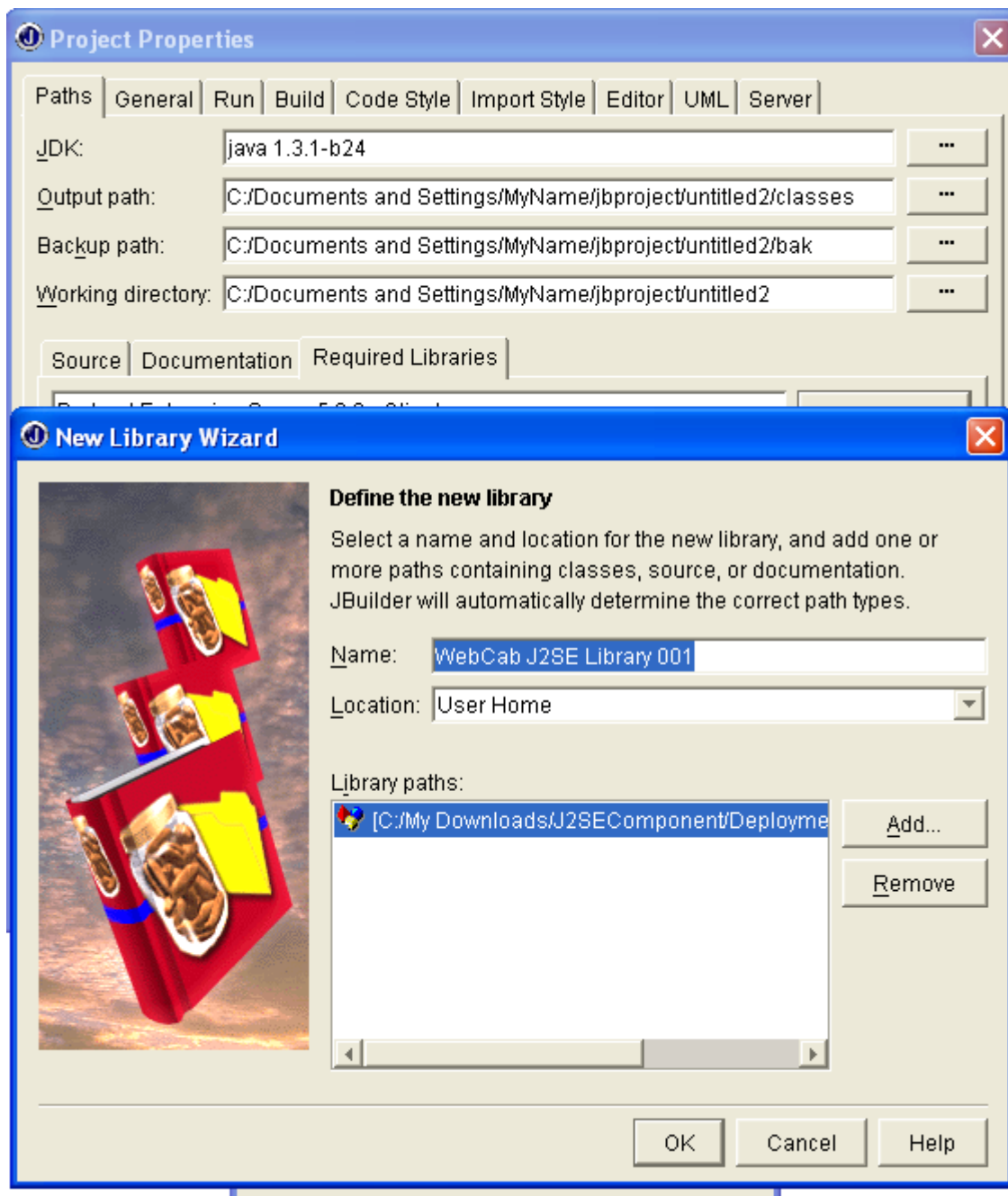
Compiling in Borland's JBuilder™

In order to use the functionality provided by our J2SE Components within your JBuilder™ projects, you will need to add the J2SE JAR file located inside the */Deployment* directory of this package to the project's list of required libraries.

Open the **Project Properties** dialog from the **Project** menu and select the **Required Libraries** tab. Click the **Add...** button, and then click the **New...** button in the newly opened dialog. Type in a distinctive name for this library and click the **Add...** button. Pick the J2SE JAR file from its location by browsing through the directory structure and press **OK**. As soon as you close each dialog by pressing the **OK** button you will be ready to compile and run your Java project together with the J2SE Components.

2.1.2 Packaging

Certain types of Java solutions require special packaging such as JAR, WAR or EAR files. For example, by packaging our JAR file within an WAR archive the JAR file may be deployed onto a Web container such as Tomcat. In order to deploy this component onto a J2EE Application server you may provide the JAR within either a WAR or EAR archive.



Adding a J2SE JAR file to the list of libraries required by a Borland JBuilder project.

Java Applets

It is always recommended that a Java Applet be archived together with its referenced class files in a Java JAR file which can then be downloaded and unpacked on the client's machine. If your Java Applet uses functionality provided by any of the classes of our J2SE Component, you will need to unpack the J2SE JAR file and repackage it with all the other files the Applet requires to run properly. In order to unpack the J2SE JAR file you will

type at command prompt:

```
jar xf J2SE-JARFile.jar
```

Then you can take the unpacked files and folders and add them to your Java Applet JAR file inputting the command:

```
jar cfM Applet-JARFile.jar <classfiles-and-resources>
```

JSP/Servlet Pages

Deployment of JSP Pages onto an Application Server is done through a special archive called Web Archive (WAR). If your JSP pages make calls to our J2SE Component, you should add the J2SE JAR file to the *WEB-INF/lib* subdirectory of the deployment WAR file. For this purpose you should create a directory named WEB-INF and a subdirectory named WEB-INF/lib, where you should put a copy of the J2SE JAR file. Then run the following command, adjusting the name of the WAR file accordingly:

```
jar uf JSP-WARFile.war WEB-INF
```

You may then deploy the Web Application as usual.

J2EE Deployment

When deploying a complete Enterprise Application which contains enterprise modules that require the presence of the classes provided by this J2SE Component, you should make sure to include the J2SE JAR file to the Enterprise Application Archive (EAR) and add a Class-path entry to the manifest file of the modules that reference this component.

First, create a file named `manifest.tmp` which contains the following:

```
Manifest-Version: 1.0
Class-Path: J2SE-JARFile.jar
an empty line
```

Then, run the following command:

```
jar ufm MODULEFile manifest.tmp
```

where `MODULEFile` is the name of the module which uses the functionality within the J2SE JAR file (for example `SessionBean.jar`). The final step is to package the EAR file as usual and run the following command:

```
jar uf J2EE-EARFile.ear J2SE-JARFile.jar
```

After performing all these steps the enterprise archive should be ready for Application Server deployment.

2.2 Testing the Component

2.2.1 Accessing an Online Demo

By far the easiest way to test the functionality of this J2SE Component is to view the online demo. The online demo can be accessed from our [J2SE Homepage](#) by clicking the '[Demo]' link corresponding to this Application.

2.2.2 Compatible Web Containers

This demo runs inside an online web container and implements the Servlet and Java Server Pages (JSP) technologies. The demo can be accessed through a web browser and is compatible with Internet Explorer 5, Netscape Navigator 4, Netscape 6, Opera 5 and higher.

2.2.3 Selecting a method to test

After clicking on the '[Demo]' link for this application from our home page the J2SE Web demo will launch within your web browser. To order to select a method from this application's J2SE component use the drop-down menu on the left hand side of the screen to navigate through all implemented functions. The menu lists all the J2SE components on the first level and their corresponding functions on the second and third level. Click on the plus icon in order to expand the J2SE component menu and left click a function to select it.

2.2.4 Inputing data

The selected function will be displayed in the center of the screen accompanied by its description and parameter characterization. Once the nature of the method is clear you may test the method by inputing the parameters within the text boxes provided or associating a database table field using our database management tool.

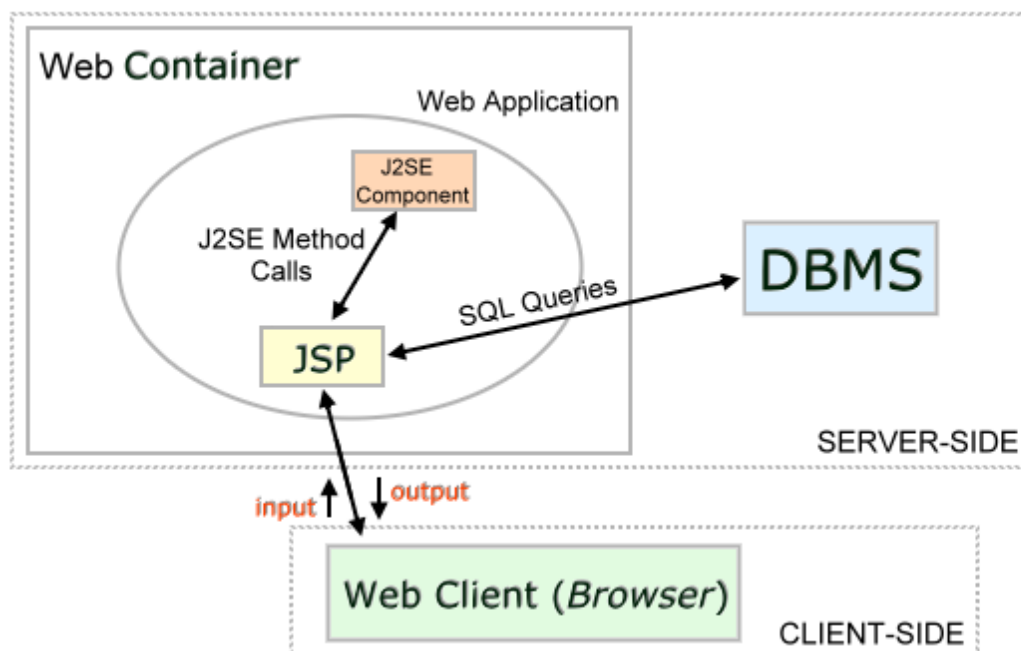
Running the demo using text boxes

In case you have chosen to input parameters by value type each number in the corresponding box while paying attention to each parameter description. When all the parameters has been input, press the "Get Result" button on the right-hand side and towards the bottom of the screen in order to request the solution from the deployed J2SE component.

If by chance any parameters are out of range you will be prompted to enter a correct value before proceeding.

Running the demo using Synthetic JDBC

If you choose to run in Synthetic JDBC mode, you will be asked to enter your DBMS connection properties, such as username and password. Next screen you will be able to choose a column for each parameter by browsing it directly from your the database. Press the the “Get Result” button to generate a report on the input columns and their corresponding computed result. The Synthetic JDBC concept is illustrated below:



www.webcabcomponents.com

Simplified exemplification of how the Synthetic JDBC works within a Web Browser.

2.3 Installing the Web Application Example Locally

This package provides an additional Web Application example which makes use of the functionality provided by this J2SE Component. This example comes wrapped inside a Java WAR file located in the */Deployment/Jsp Examples* directory of this package.

In order to test this Web Application you will first need to follow Web Container specific deployment steps, usually consisting in copying the Java WAR file to one of the Web Server's subdirectories. After deployment has completed, open a Web browser and type one of the following addresses that correspond to your Web or Application Server.

- IBM WebSphere - <http://localhost:9080/TAWebExample>
- BEA WebLogic - <http://localhost:7001/TAWebExample>

- Oracle 9iAS - <http://localhost:8888/TWebExample>
- Sun ONE Application Server - <http://localhost:81/TWebExample>
- Borland AppServer - <http://localhost:8080/TWebExample>
- Ironflare Orion - <http://localhost:8888/TWebExample>
- JBoss - <http://localhost:8080/TWebExample>

Remark This J2SE Web Application Example has also been deployed on our server. You can access it online by [clicking here](#).

Chapter 3

Technical Trading Indicators

Within this chapter we detail a range of technical indicators which each represent an trading idea which can be applied in order to develop a trading system. Our indicators we be see as falling into the following broad categories:

- [Moving Averages](#)
- [Directional Movement Indicator](#)
- [Accumulation/Distribution Indicators](#)
- [Trend or Range?](#)
- [Market Strength](#)
- [Oscillators](#)
- [Bollinger Bands](#)
- [Mean Reversion](#)
- [Stochastics](#)
- [Filters](#)

3.1 Moving Averages

Moving Averages in there various forms are used to smooth data so that the underlying trend is more discernible. Since a moving average's aim is to recognize a trending market from historical prices the sensitivity of the measures will depend on the number of historical values used. For relatively few values used the moving average may itself oscillate rapidly and give many force signals to the start of trending markets. However, the more values used within the evaluation of the moving average the further into a trending cycle before it is detected and conversely when the trend finishes or changes direction the indicator will take correspondingly longer the reflect this.

Trends always go further than rational people expect, or even imagine. Most investors don't have the stomach for extended rallies or declines. The philosophy of not having a predetermined profit objective allows us to continue with a trend for its full duration and then some. We try very hard to avoid the pitfalls of liquidating a trade too early, even at the cost of giving back large profits...

John W. Henry

Types of Moving Averages

With the construction of the moving averages themselves the main significant difference between them is the weight assigned to each price point. Simple moving averages apply equal weight to all historical prices. Exponential and weighted averages apply more weight to recent prices. Triangular averages apply more weight to prices in the middle of the time period and variable moving averages change the weighting based on the volatility of prices.

Standard Trading Systems

Within the final section we design some ways in which moving averages have been used within trading systems. Such systems are widely applicable since the existence of trending is fundamental to the technical approach to trading. The whole charting the price action (often using indicators) is to identify trends early for the purpose of trading in the direction of the trend.

3.1.1 Simple, Median and Geometric Moving Averages

Simple Moving Average

The simple (or arithmetic) x-day Moving Average (MA) of a price series is simple the arithmetic average of the closing prices over the previous x-days. That is:

$$\text{x-day Moving Average (MA)} = \frac{\sum_{n=0}^{x-1} p(n)}{x}$$

where $p(0)$ is the last closing price, $p(1)$ is the closing price on the day before and so on.

Remark The term 'moving' refers to the fact that the data used within the average is continuously updated as the asset Price move through time.

Using the Median Price

The median measure of centrality can be used with the classical moving average is order to include another filter to the historical time series. The median price is given by:

$$\text{Median Price} = \frac{\text{High} + \text{Low}}{2}$$

where High is the highest value traded on the previous day and Low is the lowest traded price. Then the x-day Median Moving Average is defined by:

$$\text{x-day Median Moving Average (MMA)} = \frac{\sum_{n=0}^{x-1} m(n)}{x}$$

where $m(0)$ is the previous days median price, $p(1)$ is the median price on the day before and and so on.

Geometric Moving Average (GMA)

The Geometric Moving Average just uses the geometric average rather than the arithmetic average as is the case with the simple moving average.

3.1.2 Weighted Moving Averages

By weighting different values within the price series within the moving average calculation you are able to assign more significance to individual and groups of values within the time series. As one might expect within the explicit implementations below the nearer dated prices have a correspondingly higher weighting the early prices.

Linearly Weighted Moving Average (LWMA)

The Linearly Weighted Moving Average (LWMA) weights the time series by assigning a weight of 1 to the oldest price and a weight of 2 to the second oldest price on so on... Until the weight of the most recent value is assigned to be the number of days in the time series. Then the LWMA is given by the sum of the weighted prices divided by the sum of the weights. That is, is the historical series has n values then the LWMA is given by:

$$\text{LWMA} = \frac{\sum_1^n P_{t.t}}{\sum_1^n t}$$

where:

- t = the time index where oldest day = 1, the second oldest day = 2, and so on
- P_t = the price of the asset on the day which is indexed by t
- n = the number of days in the original time series

Exponentially Weighted Moving Average (EWMA)

Exponential moving average is calculated by weighting today's closing price to yesterday's exponential moving average. More precisely, we choose a smoothing constant between 0 and 1, denoted by c , and then the exponential moving average is given by:

$$EWMA_t = cP_t + (1 - c)EWMA_{t-1},$$

where $EWMA_T$ is the exponential moving average on the T th day and P_t is the price of the asset on the day which is indexed by t . Hence, by induction we obtain:

$$EWMA_t = \sum_{k=0}^t c(1 - c)^k P_{t-k}$$

As k increasing the coefficient $c(1 - c)^k$, is exponentially decreasing a hence the effect of price further back in the time series rapidly decreases.

3.1.3 Variable Moving Average (VMA)

The variable moving average is an exponential moving average that automatically adjusts the smoothing weight in accordance to the volatility of the time series. The more volatile the data the more sensitive the smoothing constant used in the moving average calculation. The variable moving average was defined by Tushar Chande in an article that appeared in Technical Analysis of stocks and Commodities in March, 1992.

Most moving average calculation methods are unable to compensate for trading range versus trending markets. During trading ranges (when prices move sideways in a narrow range) shorter term moving averages tend to produce numerous false signals. In trending markets (when prices move up or down over an extended period) longer term moving averages are slow to react to reversals in trend. By automatically adjusting the smoothing constant, a variable moving average is able to adjust its sensitivity, allowing it to perform better in both types of markets.

The variable moving average (VMA) is given by:

$$VMA_t = (0.78 * VI)Close + ((1 - 0.78)VI)VMA_{t-1}$$

where VI in the volatility index (or ratio), Close is the closing price in the $(t-1)$ th period and VMA_t is the variable moving average on the t th period.

3.1.4 Kairi Indicator

The Kairi Indicator measures as a percentage of the price the divergence between the a moving average (generally the simple moving average) of the price and the price itself. The Kairi Indicator is often used with conjunction with other moving averages within trading systems.

The formulae for the Kairi Indicator is as follows:

$$\text{Kairi Indicator} = \frac{\text{MA} - \text{price}}{\text{price}}$$

where MA is the moving average being considered and price is the present price of the underlying asset.

Application

The Kairi Indicator can be used in order to take advantage of an over extended trending market. For example, in an upwardly trending market when the price gets say more than 10% above the simple moving average, the asset could be sold and repurchased when it next hits the simple moving average.

The Kairi Indicator could also be used in order to detect market tops and bottoms. The idea being that market tops and bottoms often occur when the price is at an extreme value in relation to its moving average. That is, the Kairi Indicator should take an extreme value at market tops and bottoms.

3.1.5 Moving Average Based Trading systems

All moving average based trading is based on firstly identifying and then following the trend until it changes direction. That is, a moving average system is based on the assumption that once a trend develops it is more likely to continue than to change direction.

Moving Average Crossing Trading System

This system uses the classical approach of using the crossing a moving averages to generate trading signals. We have implemented this traded signal generator within the Moving Average class/module.

Selecting the Moving Averages

You will need to select the type of moving average used and the different periods over which these moving averages are evaluated. In most, instances the simple moving average is used but in principle any type of moving average could be used. The periods of moving averages must be different. Typical choices of period used correspond roughly to convenient time periods, such as: 5 (1 week), 20 (1 month), 50 (2 months) (i.e. 50), 200 (1 year).

We will refer to the moving average with the shorter period as the Short MA, and the moving average with the longer period as the Long MA. Corresponding to the fact that they measure the trending behavior on shorter and long time spans.

Generation of Trading Signals

Trading signals are generated when:

- **Sell Signal** - Short MA crosses below the Long MA.
- **Buy Signal** - Short MA crosses above the Long MA.

3.2 Directional Movement Indicator (DMI) and Average Direction Indicator (ADX)

The Direction Movement Indicator (DMI) and Average Direction Indicator (ADX) was originally developed by Welles Wilder in order to classify the strength of price moves and trends. These ideas were originally developed into a trading system and can still be used as such. But today in more volatile markets than the ones in which these methods were originally developed these ideas are generally applied to the lesser aim of evaluating the strength of a price move or trend.

3.2.1 Comparing Ranges

The key concept within the DMI system is that of comparing the intraday price range today with that of yesterdays. In terms of the DMI system there are a number of range pair types which the system views as significant in order to evaluate the nature of the price moves or trends. In order to describe the range pair types we will use the following constance and todays and yesterdays time action:

- **todaysHigh** - the highest traded value which the asset under consideration takes during todays market action
- **todaysLow** - the lowest traded value which the asset under consideration takes during todays market action
- **yesterdaysHigh** - the highest traded value which the asset under consideration takes during yesterdays market action
- **yesterdaysLow** - the lowest traded value which the asset under consideration takes during yesterdays market action

Now the generic cases (which in fact covers all cases) we are interested in our Up Trends, Down Trends, Up Gaps, Down Gaps, Inner Range and Outer Range which are defined as follows:

Up Trend - if ($todaysHigh > yesterdaysHigh$) and ($todaysLow > yesterdaysLow$)

Down Trend - if ($todaysHigh < yesterdaysHigh$) and ($todaysLow < yesterdaysLow$)

Gap Up - if ($todaysHigh > yesterdaysHigh$) and ($todaysLow > yesterdaysHigh$)

Gap Down - if $(today'sHigh < yesterday'sLow)$ and $(today'sLow < yesterday'sLow)$

Outer Range - if $(today'sHigh \geq yesterday'sHigh)$ and $(today'sLow \leq yesterday'sLow)$

Inner Range - if $(today'sHigh \leq yesterday'sHigh)$ and $(today'sLow \geq yesterday'sLow)$

3.2.2 Positive Directional Movement (PDM) and Minus Directional Movement (MDM)

These formations are said to display either plus positive directional movement (PDM) indicating the asset is trending up-wards or minus directional movement (MDM) indicating the asset is trending down-wards. In the case of an Inner range the asset is said to display no trending tradable characteristics and in the case of the outer range the formation is tradable only if the amount the ranges extend above and below the previous days range is not equal.

Assigning Values to PDM and MDM in the different cases

The values are assigned according the following algorithm:

- **PDM**

- **Up Trend:** $PDM = today'sHigh - yesterday'sHigh$
- **Up Gap:** $PDM = today'sHigh - yesterday'sHigh$
- **Outer Range** where $(today'sHigh - yesterday'sHigh) > (yesterday'sLow - today'sLow)$, then $PDM = today'sHigh - yesterday'sHigh$
- **All other cases:** $PDM = 0$

- **MDM**

- **Down Trend:** $MDM = yesterday'sLow - today'sLow$
- **Down Gap:** $MDM = yesterday'sLow - today'sLow$
- **Outer Range** where $(yesterday'sLow - today'sLow) > (today'sHigh - yesterday'sHigh)$, then $MDM = yesterday'sLow - today'sLow$
- **All other cases:** $MDM = 0$

Remark Though the original DMI indicator uses the above means in which to evaluate the PDM and MDM, the choice of function used to measure each cases respective values could be modified while still keeping within the same ethos of the original system.

3.2.3 True Range of the Current bar (TR)

Before we can proceed to define the DMI we will need to introduce one more metric known as the True Range of the Current bar (TR). The true range extends the notion of the daily range of an asset to take into account the gaps in price between trading days. Hence, from a trading prospective the true range more accurately reflects market activity. The true range is the difference between the true low and the true high where:

- **True High:** The greater of the high or the previous close
- **True Low:** The least of the low or the previous close

Application of TR as Volatility Measure

The average daily true range (ADTR) which is often used as a measure of volatility is the simple moving average of daily true values. This measure also has natural extensions to weekly, monthly and even intraday periods.

3.2.4 Defining the Directional Movement Indicators

The plus directional indicator (PDI) and the minus directional indicator are just the PMD and MDM with respect to the True Range (TR). That is:

$$PDI = \frac{PDM}{TR}$$

$$MDI = \frac{MDM}{TR}$$

The PDI and MDI indicators like the majority of indicators are sensitive to noise within the data. In order to reduce the influence of noise and to extract the underlying trend we have implemented moving averages versions of the MDI and PDI indicators within our library. Within are implemented procedures we have used the classical arithmetic moving average and the exponential moving average approach leaving the user to specify the number of periods used. We suggest that the user first applies the PDI and MDI indicators using either 14 or 21 periods.

3.2.5 DMI Trading System

At its simplest level to DMI system states that when the PDI crosses above the MDI a buy signal is generated and when the MDI crosses above the PMI then a sell signal is generated. This however may generate an excessive number of signals and hence smoothing out each of these indicator according to a moving average will help to reduce to sensitivity of the trading system.

The central principle behind the DMI System is that when the PMI breaks above the MDI and continues to gain strength, the Bulls are firmly in control. Conversely, when

the PMI break above the MDI and continues to gain strength, the Bears are in control. Therefore, the use of a moving average actually allows us to embed systematically the "gain strength" principle within the DMI system.

Advantages to this Approach

This trading approach will reveal a trend before it is detected by most market participants. Once the trend becomes more widely recognized other market participants will tend to buy the trend and hence re-enforcing the trend dynamics. Hence the DMI system offers a good risk/reward trend following system.

Implementation of System

We implement this trading signal which can form the basis of an effective trading system within the Directional Movement Indicator class/module.

3.2.6 Directional Indicator (DX) and Average Directional Indicator (ADX)

When a trend is moving with strength, the average directional indicator (ADX) will measure the strength of the trend by measuring the spread between the PDI and MDI. The directional indicator (DX) is given by:

$$DX = 100 \left(\frac{PDI - MDI}{PDI + MDI} \right)$$

Then the ADX is evaluated by evaluating the exponential moving average of the DX over the number of periods considered. That is:

$$ADX = EMA(DX)$$

where EMA is the exponent moving average over the number of periods used. We suggest that when first apply the ADX indicator that you use either 14 or 21 periods.

3.3 Accumulation/Distribution Indicators

These indicators measure to what degree on net as asset is being accumulated (i.e. brought) or distributed (i.e. sold) by the market as a whole.

3.3.1 Accumulation/Distribution Indicator

The accumulation/distribution indicator illustrates the degree to which an asset is being accumulated or distributed by the market over a given number of periods. The indicator uses the closing price's proximity to the high or low to determine if accumulation or distribution is taking place in the market. This proximity measure is then multiplied by the

volume in order to give more weight to moves with correspondingly higher volume.

Remark We actually implement a slight generalization of the Accumulation Distribution indicator by allowing the indicator to be evaluated with respect to an ‘n’ periods rather than with respect to a single period.

Application and Trade Signal Generation

A divergence between the price action and the Accumulation/Distribution indicator can signal that a trend is nearing completion, a trends continuation and imminent break-outs from trading ranges. The actual value of this indicator is of no significance, what is significant is its change in value relative to the previous periods which can warn of a possible break-out during a trading range (falling/rising indicator), the continuation of a trend (higher highs in uptrend, or lower lows in downtrend) or a change/completion of a trend (divergence between the price action and the direction of the indicator).

3.3.2 Chaikin Oscillator

The Chaikin Oscillator (also known as the Chaikin A/D Oscillator) describes the flow of money in and up of the market. The Chaiken Oscillator presents the information contained within the A/D indicator in the convenient form of an oscillator. That is, the Chaikin Oscillator is simply the Moving Average Convergence Divergence indicator (MACD) applied to the Accumulation/Distribution Line.

Interpretation and Trade Signal Generation

A sell signal is generated when the price action develops a higher high into overbought zones and the Chaikin Oscillator diverges with a lower high and begins to fall. That is, a sell signal is generated when there is bearish divergence. Conversely, a buy signal is generated when price action develops a lower low into oversold zones and the oscillator diverges with a higher low and begins to rise. That is, a buy signal is generated when there is bullish divergence.

Remark In order to identify oversold and over brought levels, a price envelope plotted say 10 percentage above and below the exponential moving average. With this envelope an over brought region would be towards the upper end of the pricing range and a oversold level would be towards the lower end of the envelope.

The Chaikin Oscillator can also be used to time entry to existing trends by either buying the dip (when the oscillator turns down) or selling the rally (when the oscillator turns up).

Evaluation

The Chaiken Oscillator is given by:

$$EMA_3(Accumulate/Distribution) - EMA_{10}(Accumulate/Distribution)$$

where EMA_3 and EMA_{10} is the exponential moving average over 3 and 10 days respectively; and *Accumulate/Distribution* is the corresponding indicator over those 3 or 10 days respectively.

3.3.3 Chaikin Money Flow (CMF)

Chaikin Money Flow (CMF) is a volume weighted average of Accumulation/Distribution over the specified period, which is usually taken to be 21 days. The CMF offers a volume weighted indicator on the following two principles:

- The nearer the close is to the high the more accumulation is taking place.
- The nearer the close is to the low the more distribution is taking place.

Interpretation and Trade Signal Generation

A sell signal is generated in positive overbought territory when higher highs diverge into a lower high and the indicator continues to decrease. Conversely, a buy signal is generated in negative oversold territory when lower lows diverge into a high low and the indicator continues to increase.

The CMF indicator can be used as a confirmation signal after a breakout of a trading range. When a market breaks higher then the breakout is confirmed if the CMF moves into positive territory and continues to get stronger. Conversely, if the market down after a trading range then the breakout is confirmed if the CMF move into negative territory and continues to weaken.

Evaluation

The CMF indicator is evaluated for the following steps:

1. Evaluate the Volume Weighted Accumulation/Distribution over each of the days within the period considered for the calculation. The Volume Weighted Accumulation/Distribution on each day is given by:

$$\frac{(Close - Low) - (High - Close)}{(High - Low)} * Volume$$

2. Sum the Volume Weighted Accumulation/Distribution over the period and then divide the result by the sum of the volume over the period.

3.4 Trend or Range?

Within this section we consider indicators which try to determine whether a market is trading with a range or is trending. This issue is central to the development of systems which use trend following (for example a moving averages based system), or those that will trade a range (for example an oscillator based system).

3.4.1 Aroon Indicator

Within this class we define the Aroon indicator which was developed by Tushar Chande in order to establish whether a price is trending or within a trading range. The Aroon indicator is made up of the Aroon Down indicator and the Aroon Up indicator and together are said to form the Aroon indicator. We also provide the Aroon Oscillator within this component.

The determination of whether a market is trending and within a trading range is a key difficulty in the development of trading systems. The Aroon indicator has been developed in order to indicate when a trending approach such as moving averages or the trading range approach such as the application of oscillators is more appropriate.

Interpretation

Interpretation of the Aroon indicator depends on identifying one of the following three scenarios:

- **Extended Up or Down** - When the Aroon up indicator hits 100, (i.e. a high in the past day) then it could indicate the start of a up trend. If the value of the Aroon Up indicator stays in the range 70-100, and the Aroon Down indicator within the range 0-30, then it indicates that a market up trend is taking place. Conversely, when the Aroon Down indicator hits 100, (i.e. a low in the past day) then it could indicate the start of a down trend. If the value of the Aroon Up indicator stays in the range 70-100, and the Aroon Up indicator within the range 0-30, then it indicates that a market down trend is taking place.
- **Parallel Movement** - If the Aroon Up and Aroon Down are moving parallel to one another and/or lie within the range 30-70 then consolidation of the asset is indicated
- **Crossover of Indicators** - The Aroon Up indicator cross from below to above the Aroon Down indicator then a bullish signal is indicated, conversely if the Aroon Down indicator cross from below to above the Aroon Up indicator then a bearish signal is indicated.

For more information on the Aroon indicator see the article written by Tushar Chande in the September 1995 issue of Technical Analysis of Stocks and Commodities.

3.5 Market Strength

Here we discuss indicators which measure the relative strength or weakness of a market.

3.5.1 Balance of Power Indicator (BOP)

The Balance of Power (BOP) indicator, created by Igor Livshin; which captures the struggle between the Bulls and Bears throughout a trading day.

Interpretation

When the BOP indicator is towards the high of its range it will signify that the Bulls are in control, conversely when the indicator is towards the lows of its range it signifies that the bears are in control. If the indicator moves from a high positive range to a lower positive range it signifies that the buying pressure is decreasing. Conversely, if the indicator moves from a low negative range to a higher negative range it signifies that the selling pressure is decreasing.

Evaluation

The BOP indicator is evaluated by the following formulae:

$$\text{BOP} = \frac{(\text{Close} - \text{Open})}{(\text{High} - \text{Low})}$$

where close is the day's closing price, open is the day's opening price, high is the highest traded price during the day and low is the lowest traded price during the day

3.6 Oscillators

Within this section we deal with Oscillators such as the money flow index, stochastics, momentum and rate of change (ROC) indicators. Oscillators are generally used to identify short term price reversal points as opposed to longer term trending dynamics.

3.6.1 Money Flow Index (MFI)

The Money Flow Index (MFI) measures the strength of money flowing in and out of a security. The MFI is related to the Directional Movement Indicator of William Wilder, but the MFI system also takes into account the volume as well as the price action.

Interpretation

Divergence of the indicator within a continuing trend indicates that a reversal is imminent. The MFI also is a good means to signal market tops and bottoms. A reasonable rule is a market top is given more significance when the MFI is above 80 and a market bottom is given more significance when the MFI is below 20.

Evaluation

The Money Flow Index (MFI) is evaluated over a given user defined number of trading periods. The evaluation of the MFI follows the below steps:

1. Evaluates the Money Flow on each day. The Money Flow is the product of the Typical Value Indicator (see Filters) for that day and the volume.

2. Each day is either classified as either a Positive Money Flow day, Negative Money Flow day or no Flow day. If the typical price increases then the day is said to be a Positive Money Day, if it decreases then it is said to be a negative money day and if it stays the same then it is said to be a no flow day.
3. The Positive Money and Negative Money Flow's are calculated by summing the Money Flow over the Positive Money Flow Days or the Negative Money Flow days respectively.
4. The Money Flow Index (MFI) is evaluated by dividing the Positive Money Flow by the Negative Money Flow.

3.6.2 Momentum and Rate of Change (ROC) Indicators

The Momentum and Rate of Change (ROC) will get similar numerical values and can be used together or interchangeable within a system.

The momentum indicator as the name suggests is the velocity with which the price is rising or falling, and hence will reflect how aggressively the asset is being purchased or sold. Whereas the ROC indicator approximately represents percentage change of the asset over the considered period.

Interpretation

Extended values and/or turning points of the momentum are good indicators of oversold or overbought conditions (respectively).

3.7 Bollinger Bands

Bollinger Bands are a type of envelope that are plotted at standard deviation levels above and below the corresponding moving average. This produces an effect of having the bands widen during periods of higher volatility and contract during less volatile periods. Bollinger Bands indicate the relative supply and demand for a given asset. If the asset moves close to the top of the envelope then it indicates that there is strong demand for the asset, conversely if the asset hugs the bottom of the trading range then it indicates that there is oversupply of the asset.

Since the Bollinger Bands will nearly always be combined with other indicators when forming a trading system the number of periods used in the evaluation of the standard deviation of the stocks and the moving average will vary. For those without design restrictions a popular choice of the number of time periods is around 20-23.

What we Implemented

We provide methods from which the upper and lower Bollinger Band can be evaluated from the time series, the number of periods used and the number of standard deviation shifts used to create the bands.

3.8 Mean Reversion

Some market times series such as interest rates, commodity prices, FX rates, and implied volatility of stock and index options are known to exhibit the property of mean reversion. Within this section we detail the indicators which in some way or another rely on the fact that many market time series will revert to there mean.

3.8.1 Commodity Channel Index (CCI)

The Commodity Channel Index (CCI) developed by Donald Lambert, measures the variation of a security's price from its statistical mean. High values show that prices are unusually high compared to average prices whereas low values indicate that prices are unusually low. The CCI can be used effectively on any type of security, but clearly it is most applicable where the security has should a strong degree on mean reversion.

Lambert originally commended that the CCI was designed to capture the trade cycle (i.e. low-to-low or high-to-high) turns in commodity markets. The system assumes that commodities move in cycles and uses 1/3 of the cycle period for the evaluation of the CCI. We allow the uses to specify the length of the calculation period used but we advise that you take the calculation cycle to be approximately one third of your estimate for the length of the trade cycle.

Calculation Period: the number of days used in the evaluation of the CCI. In Donald Lambert's original system this was taken to be one third of the estimate length of the trade cycle.

Remark Within the evaluation procedure (step 2), we multiply the result by the constant 0.015. The constant was originally used in Lambert's system and has been found to ensure that around 70-80 percent of all the values given by the CCI lie the range $[-100, +100]$. Hence, the constant is just used to calibrate the indicator with respect to the range $[-100, +100]$.

Interpretation of the CCI Indicator

Significant signals are generated when either the CCI starts to diverge from the price action which will signify a correlation in the price, or when the CCI extended (typically above 100 or below -100) which indicates oversold or overbought conditions.

Further details concerning the CCI can be found in an article by Donald Lambert that appeared in the October 1980 issue of Commodities (now known as Futures) Magazine.

3.9 Stochastics

The Stochastics Oscillator compares the closing price with the price over a given period. The rationale is that if the closing price is near to the highest traded price over a given number of previous sessions then the stock is bullish. Similarly, if the closing price is near lowest traded price over a given number of previous sessions then the stock is bearish.

3.9.1 Interpretation and Application

Stochastic Oscillator's produce two time series, %K, and its moving average (MA) usually denoted by %D. These two lines are usually plotted on the same graph since the interaction is generally used in order to determine trading signals.

Below we describe three popular ways in which the Stochastic indicator is interpreted in order to produce trading signals:

1. **Extreme Values** - Buy when the Oscillator (either the Stochastic %K or its moving average %D) falls below a specific level (e.g., 20) and then rises above that level. Sell when the Oscillator rises above a specific level (e.g., 80) and then falls below that level. This approach is the preferred method of the Stochastics original creator George Lane.
2. **Crossing** - Buy when the Stochastic %K, crosses above the MA %D and sell when the Stochastic %K falls below the MA of Stochastic %D. This will give more trading signals than the above method and is only suitable when the market is trending over the range that you are considering.
3. **Divergence** - By searching for any divergences between the price dynamics and the Stochastic indicator %K. The Stochastic will often indicate when a trend is about to change direction (i.e. "its losing steam"). A good example is when the price is making a series of higher highs but the Stochastic is falling to make higher highs.

Remark If the fast Stochastic whip-saws excessively the user of this indicator may wish to compare two %D Stochastics in a similar fashion. Where the first moving has a lower period and hence is more volatile than the two moving average which has a longer period.

3.9.2 Evaluation

Evaluation the fast %K Stochastic

The (fast) Stochastic %K depends on the following variable:

1. **Periods** - the number of previous time periods used over which the closing price is compared.

Now the formulae for the Stochastic %K is:

$$\left(\frac{LastClose - \text{Lowest low over periods}}{\text{Highest high over periods} - \text{Lowest low over periods}} \right) \times 100$$

where the "lowest low" (respec. "highest high") is the highest (respec. lowest) close of the asset over the period under consideration. Since the "Last close", will lie between the highest high and lowest low this indicator will lie between 0 and 100.

Evaluation of the %D Stochastic

As one would expect the Moving Average %D of the Stochastic depends on:

1. **%D Periods** - The number of periods over which the moving average is evaluated.
2. **MA Method** - This is the method by which the moving average of the Stochastic %K is evaluated. We offer a range of methods for the evaluation of the moving average, these include: simple, median, exponential, geometric, weighted, linearly weighted and exponentially weighted.

Remark When the simple moving average is used and it is evaluated over 3-days, then the % D Stochastic reduces to what is known as the slow % K Stochastic.

Once the parameters are set the %D Stochastic is given by:

MA over the specified period

That is, if the simple moving average is chosen and the period is set to be 5 days over which the %K Stochastic was 63, 74, 73, 79, 83; then:

$$\%D \text{ Stochastic} = \frac{63 + 74 + 73 + 79 + 83}{5}$$

3.10 Filters

We include a number of filters which can be applied to 'clean' the underlying time series making it more amenable to analysis.

At present we offer the following filters:

1. **Typical Price** - The Typical Price Indicator is arithmetic average of the high, low and closing price for a trading day. The Typical Price is often used in place of the closing price within trading systems.
2. **Median Price** - The Median Price Indicator is the midpoint of each days trading range. By replacing an interval with a point you are able to draw a daily line chart of a securities price action.

Chapter 4

Programmer's Guide

This chapter describes development techniques for various business solutions that make use of the functionality provided by our J2SE™ Component. We analyze both standard and enterprise Java solutions including stand-alone Java applications, Java Applets, JSP pages and Enterprise JavaBeans™.

4.1 Developing with J2SE

The J2SE Edition of this product offers core-level functionality to the Java developer allowing the implementation of fully personalized components and applications for specific business problems on a variety of deployment environments. Irrespective of the complexity of the project a programmer may be working on, this J2SE Component can be integrated in an equally straightforward manner by providing the needed functionality in a compact and precise way.

The Technical Analysis component comes wrapped inside a Java JAR file called TAJ2SE.jar. This class library contains all classes that provide the functionality offered by this J2SE Component as documented inside the API reference directory of this package (*/JavaDocs*). Depending on the type of Java solution you are developing, you will be using this JAR file in a specific way. Once you have chosen a particular implementation of a deployment framework, the structure of your source code will need to adapt accordingly. In the following discussion we describe several basic J2SE implementation examples that can be used to make use of our product's functionality.

4.1.1 Stand-alone Java Applications

A stand-alone Java application is a Java class which acts as a client for any type of Java components ranging from JavaBeans, class libraries and EJB components. The source code listed below defines a standard stand-alone application skeleton which communicates with a class within a J2SE Component. By creating an instance, calling a method, sending in a set of parameters and then retrieving the returned result of the method call. As soon as the method call has gone through successfully the method continues by displaying its

result on the screen.

```
/*
 * The class we are using is located in the webcab.lib.j2sepackage package.
 */
import webcab.lib.j2sepackage.*;

public class StandAloneExample {

    public static void main (String[] args) {

        /*
         * Creates an instance of the J2SEClass class
         */
        J2SEClass instance = new J2SEClass ();

        /*
         * Defines the method parameter.
         */
        double parameter = 25;

        /*
         * Invokes a method with the specified parameter
         * and puts the result in the result variable.
         */
        double result = instance.computeResult (parameter);

        /*
         * Prints out the retrieved result.
         */
        System.out.println ("Method 'computeResult' in class J2SEClass
                             returned " + result + ".");
    }
}
```

If the method `computeResult` of the `J2SEClass` class returned 100, this stand-alone application would print the following:

```
Method 'computeResult' in class J2SEClass returned 100.
```

4.1.2 Java Applets

Applets run within a Java enabled web container allowing real-time user interaction on the Internet. The source code listed below defines an Applet which performs the same calculations as the stand-alone example above and then graphically displays the result.

```
/*
 * The class we are using is located in the webcab.lib.j2sepackage package,
 * while the Applet specific classes reside in javax.swing.
 */
import webcab.lib.j2sepackage.*;
import javax.swing.*;

public class AppletExample extends JApplet {

    public void init () {

        /*
         * Creates an instance of the J2SEClass class
         */
        J2SEClass instance = new J2SEClass ();

        /*
         * Defines the method parameter.
         */
        double parameter = 25;

        /*
         * Invokes a method with a specified parameter
         * and puts the result in the result variable.
         */
        double result = instance.computeResult (parameter);

        /*
         * Prints out the retrieved result.
         */
        getContentPane ().add (new JLabel ("Method 'computeResult' in
            class J2SEClass returned " + result + "."),
            java.awt.BorderLayout.CENTER);
    }
}
```

This Applet source code will be compiled and placed together with all classes from the TAJ2SE.jar file in another Java JAR file and be deployed inside a web container. Users connecting through a web browser will first download this newly created Java JAR file and then interact with the Applet's graphical interface.

4.2 J2EE™ Solutions Using J2SE Components

The Java™2 Enterprise Edition platform provides a great variety of business framework solutions, which transfer hard-coding to the Application Server providers and enables the developer to exclusively concentrate on the business problem. In the following sections we discuss two of the most popular J2EE technologies, Java Server Pages (JSP) and Enterprise JavaBeans (EJB), which are universally supported by all Application Server providers.

4.2.1 Writing JSP Pages

JSP Pages are very much like Applets since they both basically offer the same functionality to the end-user, interaction within a web page. But even though they both display their results in a browser, the main difference is the fact that JSP pages are server-side applications, which never get downloaded onto the client's machine and never use local CPU or memory resources. Yet as previously stated the code you will write when designing your JSP pages to use our J2SE Components will be very similar to client side applications we have already described. That is, you will need to make the following steps:

- Create an instance of the class you wish to use
- Make a method call by sending the required parameters
- Display the result in HTML form

The following JSP page exemplifies this process:

```
<!-- The class we are using is located in the webcab.lib.j2sepackage package.
-->
<%@page import="webcab.lib.j2sepackage.*"%>
<HTML>
<HEAD><TITLE>JSP using J2SE Component Example</TITLE></HEAD>
<BODY>
<%
    /*
     * Creates an instance of the J2SEClass class
     */
    J2SEClass instance = new J2SEClass ();

    /*
     * Defines the method parameter.
     */
    double parameter = 25;

    /*
     * Invokes a method with the specified parameter
     * and puts the result in the result variable.
     */
    double result = instance.computeResult (parameter);
%>
<!-- The JSP page displays the result inside the web browser. -->
Method 'computeResult' in class J2SEClass returned <%=result%>.
</BODY>
</HTML>
```

The JSP page will be compiled into an HTML page by the web container on the server and be served to the client's machine where the end-user will be able to read the computed result, as if it had been run locally.

4.2.2 Designing EJB™ Components with J2SE Components

Enterprise JavaBeans™ are server-side components which provide the same business functionality as J2SE Components while not taking up local system resources. Developers writing EJB components that use the functionality provided by J2SE Components will need to include the TAJ2SE.jar file with their enterprise deployment archive (EAR) and perform local calls like shown in the previous examples. The following source code defines the implementation class of a stateless Session Bean which makes calls to a J2SE class.

```
/*
 * The class we are using is located in the webcab.lib.j2sepackage package,
 * while the EJB specific classes reside in javax.ejb.
 */
import webcab.lib.j2sepackage.*;
import javax.ejb.*;

public class EJBClassExample implements SessionBean {

    /*
     * EJB specific methods.
     */
    public void ejbActivate() throws RemoteException {}
    public void ejbPassivate() throws RemoteException {}
    public void ejbRemove() throws RemoteException {}
    public void setSessionContext (SessionContext c)
        throws RemoteException {}

    /*
     * Business enterprise method.
     */
    public boolean checkResult (double parameter) {

        /*
         * Creates an instance of the J2SEClass class
         */
        J2SEClass instance = new J2SEClass ();

        /*
         * Invokes a method with the received parameter
         * and puts the result in the result variable.
         */
        double result = instance.computeResult (parameter);

        /*
         * Returns true if the result is positive, false otherwise.
         */
        return (result >= 0);
    }
}
```

Together with its home and remote interfaces this class will encapsulate a stateless Session EJB that serves client-side requests by making local J2SE calls.

Remark For further details concerning the use of this component as an EJB, please see the Programmers Guide of the J2EE Editions PDF documentation.

4.3 Support for Developers

4.3.1 Compilation and Custom Modification of Source Code

This J2SE Components source files have been compiled using Sun's J2SDK1.4.2, should you prefer compilation of the source code using another compiler then please contact us. Some functions (for example standard mathematical operations) are embedded within the source code, if you license classes which you believe offer a similar level of functionality from a third party and wish to include these classes within our components then please contact us. Both of the above services are offered free of charge at WebCab's discretion.

4.3.2 Online

Should you have any questions or queries, please feel free to contact us via our support forum at:

<http://www.webcabcomponents.com/support/index.php>

For updates and new releases, visit:

<http://www.WebCabComponents.com/Java/index.shtml>

Chapter 5

Examples

Within this chapter we illustrate how the Technical Analysis Component can be applied to solve real life problems. We provide with this Component examples of two types:

1. **QA Examples**
2. **Custom Examples**

5.1 Question and Answer (QA) Client Examples

5.1.1 Overview

Within this folder we offer Java client examples for the J2SE Business Components provided within this package. In particular, for each business method contained within each business Component we provide a client side example; which illustrates how functionality contained within this component can be applied to solve real world problems. In addition, to these examples we also include custom applications which solve some of the generic problems which this Component is designed to address.

5.1.2 Structure of QA Examples Directory

By drilling down the QA Client directory structurr you will find the following six directory levels:

1. Client Level
2. Technology Level
3. Business Module Level
4. Business Class Level
5. Example Level
6. Custom Example Level

which have the structure as shown in the following diagrams.

5.1.3 Top Four levels of the QA Directory Structure

```

                (Technology Level) (Business Module Level)          (Business Class Level)

Client +
      |
      + Java -----+
      |              |
      + Readme.txt   + 'Business Module 1' -----+
                      |                               |
                      + 'Business Module 2' --+... + 'Business Class 1' -----
                      |                               |
                      + QALibrary.jar             + 'Business Class 2' --+...
                                                  |
                                                  + ...

```

5.1.4 Bottom Two levels of the QA Directory Structure

```

                (Example Level)                (Custom Example Level)

-----+
      |
      + 'BusinessClass'.java
      |
      + compile.cmd (.sh for Linux)
      |
      + run.cmd (.sh for Linux)
      |
      + run_gui.cmd (.sh for Linux)
      |
      + 'CustomClient' -----+ 'CustomClient'.java
      |                       |
      + ...                   + compile.cmd (.sh for Linux)
                              |
                              + run.cmd (.sh for Linux)

```

5.1.5 Quick Start Guide

Browse down to the particular client within the business module for the given client technology you are interested in. Run the compile.cmd (.sh for Linux), script in order to compile the example and then run the run.cmd (.sh for Linux), script in order to run the example.

5.1.6 Explanation of the QA Directory Structure and its files

1. Client Level

The directory containing the Questions and Answer Examples is named 'QAClients'. This directory can be located by using the Index.html file, which is presented at the end of the installation process, by selecting:

Run Examples > Client Examples Directory

Within this folder you will also find the following file:

- Readme.txt - this readme file

2. Client Technology Level

Within the 'Client' folder is the Technology Level of the QA directory structure. Within this folder you will find sub-directories; which contain examples written in each of the Java Client Technologies in which the clients have been implemented.

As mentioned above within the overview we provide an implementation of each 'Question and Answer (QA)' example using each of the client technologies used. Selecting one of these folders in order to view the client implemented within the corresponding technology.

3. Business Module Level

After selecting one of the technology sub-folders at the 'Client Technology Level' (see (2) above) you will enter a folder which contains a sub-folder for each of the modules contained within this component package. The modules are convenient collections of business classes containing the business functionality offered by this component. Each module has a sub-folder which contains the 'Question and Answer' examples of each method of the classes which it contains.

This directory will also contain the following file:

- QALibrary.jar - contains utility functionality necessary for running the clients.

4. Business Class Level

At the business class level under the module level we are presented with one sub-folder for each of the classes within the business module. Each of these folders contains the files of the examples for each of the methods contained with the respective class.

5. Example Level

At the Example level you will be presented with the files which allow you to compile and then run the examples of the application of the Business methods of the respective class. The files within this directory which constitute the Client example are as follows:

- Source Code File(s) - Depending on the Java compatible technology selected at the 'Technology Level' this folder will contain a source code file(s) of the client.
- Compile Batch File - Assuming that you have access to the relevant compiler the compile batch file (compile.cmd or compile.sh) once run will compile the client example from the folders source code file. Below we include some remarks on how to obtaining suitable Java compiler in order to be able to compile and then run the Java examples.
- Run Batch File - The run batch file (run.cmd or run.sh) will run the examples executable file (exe) which is generated from the compilation of the source code file. If the example over runs your console screenful then press space in order to page down or enter is order to scroll down.

6. Custom Example Level

At the Custom Example level you will find a source code file containing the implementation of the Application which addresses a typical question for which the Component is designed. The source code has a linear structure with full inline commentary. By just running the compile and then run scripts contained within this directory you will be able to solve to given problems view the results obtained. The files contained within each Custom Client Example directory are as follows:

- Source Code File(s) - The custom clients source code in the appropriate client technology.
- Compile Batch File - Assuming that you have access to the relevant compiler the compile batch file (compile.cmd or compile.sh) once run will compile the client example from the folders source code file.
- Run Batch File - The run batch file (run.cmd or run.sh) will run the examples executable file which is generated from the compilation of the source code file. If the example over runs your console screenful then press space in order to page down or enter in order to scroll down..

5.1.7 Remarks on Java compilers

1. Required compilers and how to test for them

In order to compile the Java examples you will require a suitable Java compiler which should be installed and configured within your environment variable PATH.

The easiest way in which to test whether you have access to a Java compiler at your command prompt is to type the follow command at the command line:

java - Invokes the Java compiler

2. How to obtain the required Java compiler

Compilers for the Java programming language are provided within Sun's Java SDK which can be obtained from:

<http://java.sun.com>

5.2 Custom Examples

5.2.1 Database Example with JDBC Mediator

The Database Example is located inside the *Client/Indicators/SimpleTradingSystem* directory. The following steps are required before running the Database example.

1. Installing our Tables (MySQL Only)

In order to create the database structure from which you are able to load the output results, you will need to run the 'create.sql' scripts in the 'Data' subdirectory. Open the MySQL prompt from the 'Data' subdirectory and:

- Select (or create and then select) a database you can spare for tables named AMD, BEAS, DELL, IBM, MSFT, ORCL, and RATE. For example, if you have decided using the 'test' database, you would optimally type the SQL command to create it (unless it already exists):

```
CREATE DATABASE test;
```

and then, you would type the following to select it:

```
USE DATABASE
```

- Run the 'create.sql' SQL script located in the 'Data' subdirectory of the current directory by typing at the same MySQL prompt the following:

```
source create.sql
```

If this fails, make sure you have started the MySQL prompt from the 'Data' subdirectory (this is where the database files for this example are stored).

2. Configuring the Database Connection

Edit the Java source code file by filling out JDBC information about your database, such as:

- (a) Driver Name (e.g. `com.jdbc.mysql.Driver`)
- (b) JDBC Url (e.g. `jdbc:mysql://localhost/test`)
- (c) Username and Password

If you are unsure whether you have a JDBC Driver for your Database, skip this step and try running the example with the predefined values. If that fails, you will need to probably download the latest driver.

3. Running the example

Run the ‘compile’ script (`compile.sh` for Linux, `compile.bat` for Windows) to compile the Java source code, and then the ‘run’ script to see the results.

Uninstalling the Source Data

To delete all the tables used in this example, open the MySQL prompt from the ‘Data’ subdirectory, select the database where you created the tables, and run the following:

```
source delete.sql
```

Chapter 6

Guide to WebCab Components

6.1 The Company

WebCab is a privately owned British company that has built business solutions since its inception in 1999. We continue to refine and develop our Mathematical and Financial Framework which we have implemented within the J2SE, J2EE and .NET platforms.

6.2 Presentation of Products

We aim to provide good quality, useful information to help you decide which J2SE Component best suits your development needs. WebCab is committed to honesty and realism when presenting our products. In order to achieve this a detailed, clear and factual style for presentation is adopted within our documentation and marketing material.

6.3 Supported Clients, IDEs, Containers and DBMSs

We support all major development, server and client side technologies. Each product contains detailed examples and advice concerning the integration of our components within existing infrastructure. Our documentation provides the information that the developer needs to get their applications up and running as quickly and as easily as possible.

We detail exactly how the developer can use our J2SE Components with the following technologies:

- Client containers - Internet Explorer, Netscape Navigator, Mozilla, Opera
- IDEs - JBuilder, BEA Studio, Sun ONE (formerly Forte For Java), Eclipse, Oracle JDeveloper
- Client Side Technologies - Application Clients, Frames, Applets
- DBMS - Oracle, IBM DB2, SQL Server, Sybase, MySQL

- Platforms - Windows XP/2000/NT, Linux, Solaris, IBM AIX, HP-UX

For these technologies we include all the installation scripts and template examples which will help the developer quickly and easily assemble their multi-tier enterprise application.

6.4 Transparent Functionality

All technical and business intelligence incorporated within our products is described within the associated documentation. This allows the developer to see the nature of the methodology implemented within our proprietary algorithms.

6.5 Code Conventions

WebCab adheres to the ‘Code Conventions for the Java Programming Language’ as specified on April 20, 1999 by Sun Microsystems Inc. These conventions cover filenames, file organization, indentation, comments, declarations, statements, white space, naming conventions and programming practices. Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.

6.6 Company Culture and Activity

WebCab Components is focused solely on the production of high quality software modules within our Mathematical and Financial Framework. The WebCab team contains a wide range of expertise and experience within the academic, investment banking and software development worlds.

Within the company there exists significant internal competitive pressures with constant peer review and evaluation, resulting in higher quality products, which adhere to high professional standards and offer extended functionality.

6.7 Product Life cycle

We continuously add value to our products by evolving them according to new J2SE specifications, customer feedback and market demands. We give particular emphasis to incorporating our clients suggested modifications and additions into our product development cycle.

6.8 Support, Warranty and Upgrades

WebCab warrants that each component will perform substantially in accordance with the accompanying written material. We provide with all our products without charge sixty

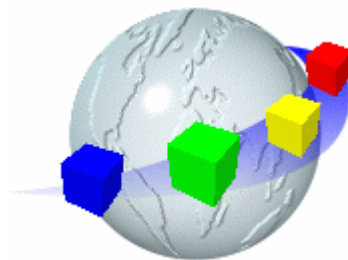
(60) days product support services including fixing bugs, compatibility issues and other technical support issues.

All maintenance updates (including service packs) will be distributed free of additional license cost. New version releases of this product will be offered to present licensee holders at a greatly reduced rate.

Dr Ben Fairfax

Founder and CEO

WebCab Components - From Developer, To Developer



Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.