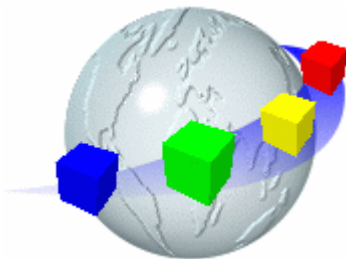


WebCab Bonds

WebCab
Components



Preface

This documentation accompanies the WebCab Bonds J2EE Application. WebCab Bonds contains methods which enable the implementation of models for the pricing and risk analysis of interest rate instruments. In particular we cover fundamental topics from the theory of bonds, fixed interest bonds and interest based calculations.

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 by detailing deployment techniques on the most widely used application servers. The third, fourth and fifth chapters contain the mathematical documentation, which represents the theoretical background of this component's implemented features. Chapter six contains the programmer's guide containing a road map for developers to take advantage of every feature and capability. In chapter seven we provide examples which illustrate how features detailed within the programmer's guide can be applied in practice. In chapter eight, UML diagrams are provided for each of the products interfaces. Finally, we introduce WebCab Components, its philosophy and approach to serving the JavaTM development community with robust and powerful Applications.

If you have any suggestions for additional functionality, which we could include into future development cycles or any other queries then please feel free to contact us via our support forum 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	4
1.3 Prerequisites and Compatibility	4
1.3.1 System Requirements	4
1.3.2 Compatibility	5
2 Where do I Start?	6
2.1 Self-Deploy Installer	6
2.2 Deploying this Component	7
2.3 IBM WebSphere Application Server	7
2.3.1 Deploying on IBM WebSphere V4.0 and V5.0	7
2.4 BEA WebLogic Server	9
2.4.1 BEA WebLogic Server 6.1	9
2.4.2 BEA WebLogic Server 7.0 (J2EE 1.3 compliant)	10
2.5 Oracle9i Application Server	12
2.5.1 Oracle9i v9.0.2	12
2.5.2 Oracle9i v9.0.3 (J2EE1.3 certified)	13
2.6 Sun ONE AppServer 7	14
2.7 Borland Enterprise AppServer 5	14
2.8 Sybase EAServer 3.6	16
2.9 Ironflare Orion Server	16
2.9.1 Orion 1.5.2 (and Orion 1.5.4)	16
2.9.2 Orion 1.6 (EJB2.0 compliant)	17
2.10 JBoss Open Source Application Server	18
2.10.1 JBoss 2.4.4 (and JBoss 2.4.3)	18
2.10.2 JBoss 3.0.0 (EJB2.0 compliant)	18
2.11 Testing the Component	19
2.11.1 Accessing an Online Demo	19
2.11.2 Compatible Web Containers	19

2.11.3	Selecting a method to test	19
2.11.4	Inputing data	19
2.12	Using the Web Demo	20
3	Fundamental Theory of Bonds	21
3.1	Types of Rates and pricing	21
3.1.1	Types of Rates	21
3.1.2	Bond Pricing	22
3.1.3	Zero Rates, Bootstrapping and the Constructing the Zero Rate Curve	23
3.2	Forward Rates and Forward-Rate Agreements	24
3.2.1	Evaluating the Forward Rate	24
3.2.2	Forward-Rate Agreements	25
3.3	Duration	26
3.3.1	Price and Duration of a Bond	26
3.3.2	Duration of a Bond Portfolio	26
3.3.3	Rescaling the Duration	27
3.3.4	Duration in a general setting	27
3.3.5	Duration Based Hedging and Convexity	27
4	Yield of Fixed-Interest Bonds on Interest Payment Dates	30
4.1	Simple Yield to Maturity (for Japanese Government Bonds (JGB))	30
4.2	Gross Redemption Yield and Current Price	31
4.3	Net Redemption Yield (excluding CGT)	32
4.4	Holding Period Return (US and UK)	33
4.5	Bisection and Newton Raphson	34
4.5.1	Bracketing One Dimensional Equations	34
4.5.2	Bisection Method	35
4.5.3	Newton-Raphson Method	35
5	Interest Calculations	37
5.1	Preliminaries and Notation	37
5.1.1	Preliminaries	37
5.1.2	Notation	38
5.2	Simple Interest	39
5.2.1	Basic formula	39
5.2.2	Real worth	39
5.2.3	Real return	40
5.3	Compound Interest	40
5.3.1	Accumulated values	40
5.3.2	Real worth	40
5.3.3	Real return	41
5.3.4	Depreciation	41
5.3.5	Force of interest	42
5.4	Effective and Nominal Interests	42

5.4.1	Effective rate of interest	42
5.4.2	Real return	43
5.4.3	Nominal rate of interest	43
5.5	Net Present value	43
5.6	Annuities	44
5.6.1	Accumulated Values of an Annuity	44
5.6.2	Present Value of Annuities	45
5.6.3	Series of multiple payments per interval in arrears	46
5.7	Yield (internal rate)	46
5.7.1	Finding the yield	46
5.7.2	The relationship between real and nominal yield	47
5.8	Repo agreements	47
6	Programmer's Guide	48
6.1	Introduction to Fixed-Interest Bonds Component	48
6.2	Client-Side Implementation Procedure	49
6.3	Disposing Resources	50
6.4	Using JDBC Mediator with our EJBs	50
6.4.1	Overview	50
6.4.2	Connecting to your Database Server	51
6.4.3	JDBC Components	53
6.5	Support for Developers	55
6.5.1	Compilation and Custom Modification of Source Code	55
6.5.2	Online Support	56
7	Examples	57
7.1	Question and Answer (QA) Client Examples	57
7.1.1	Overview	57
7.1.2	Structure of QA Examples Directory	57
7.1.3	Top Four levels of the QA Directory Structure	58
7.1.4	Bottom Two levels of the QA Directory Structure	58
7.1.5	Quick Start Guide	58
7.1.6	Explanation of the QA Directory Structure and its files	59
7.2	Custom Examples	60
7.3	Fundamental Theory of Bonds	60
7.3.1	Zero Rates	61
7.3.2	Forward Rates	61
7.3.3	Treasury Price	63
7.3.4	Duration	66
7.3.5	Database Example with JDBC Mediator	71
7.4	Yield of Fixed-Interest Bonds on Interest Payment Dates	72
7.4.1	Gross Interest Yield	72
7.4.2	Gross Redemption Yield	73
7.4.3	Holding Period Return	73

7.5	Interest Investments	73
7.5.1	Compound Interest	73
7.5.2	Effective Rate of Interest	74
7.5.3	The accumulated value	74
7.5.4	Present Values	74
7.5.5	Yield	75
7.5.6	Repo agreements	75
8	Bonds UML Diagrams	76
8.1	Introduction and Key	76
8.2	UML Diagrams	77
8.2.1	CalculatingZeroRates	77
8.2.2	DurationConvexity	77
8.2.3	FixedInterestBonds	78
8.2.4	ForwardRates	78
8.2.5	NoSolutionException	79
8.2.6	TreasuryPrices	79
9	Guide to WebCab Components	80
9.1	The Company	80
9.2	Presentation of Products	80
9.3	Supported Clients, IDEs, Containers and DBMSs	80
9.4	Transparent Functionality	81
9.5	Code Conventions	81
9.6	Company Culture and Activity	81
9.7	Product Life cycle	81
9.8	Support, Warranty and Upgrades	82

Chapter 1

Introduction

1.1 Product Description

1.1.1 Overview

Model the pricing and risk analytics of interest rate cash and derivative products. We cover the fundamental theory of bonds including: Treasury bonds, Yield/Pricing, Zero Curve, Forward rates/FRAs, Duration and Convexity. We also cover the topics of Fixed-Interest bonds and interest based calculations.

1.1.2 Details

WebCab Bonds implements the following functionality:

Fundamental Theory of Bonds

- **Pricing and Yield**
 - **Pricing** - Discounted cash flows model in accordance with the risk free interest rate
 - **Yield to Maturity (YTM)** - the YTM (also known as the Internal rate of Return (IRR) can be evaluated for any bond where the market price and the coupon payments until maturity are known.
 - **Treasury Price** - evaluate the price of a Treasury bond from the Treasury zero rates.
 - **Bond Yield** - returns the yield of a Treasury bond when the price and coupons are known.
 - **Par Yield** - we provide methods for calculating the Par Yield where the number of yearly payments and the annuity may vary.
- **Constructing the Zero Rate Curve** - using the technique known as bootstrapping and linear interpolation we are able to construct the zero rate curve.

- **Forward Rates and FRAs**

- **Evaluation of Forward Rates** - the forward rate for a given period can be evaluated from the zero rates at the start and end of that period.
- **Forward Rate Agreements (FRAs)** - we provide a method which shows to value of a FRA and the cash flows when the contract is settled.

- **Duration and Convexity**

- **Duration** - the Duration of a bond, bond portfolio, interest rate future and the rescaling of Duration according to different interest compounding conventions.
- **Duration based hedging** - Duration-Hedge Ratio, Convexity and its use in hedging interest rate risk.

Yield of Fixed-Interest Bonds on Interest payment dates

- **Simple Yield to Maturity** - As used in Japanese bond markets to calculate the yield to maturity (simple yield to maturity) rather than the usual compound interest method (redemption yield).
- **Gross Redemption Yield** - For an interest payment date the gross redemption yield is given. We follow the convention in the US and UK to calculate and express redemption yield as a yield per annum, convertible half-yearly.
- **Net Redemption Yield** - The gross redemption yield on an interest payment date taking into account the investors income tax position.
- **Holding period return** - The yield over the period the stock was held by the investor according to US and UK interest payment conventions.
- **Rate of Payments** - Knowing the series of payments of one per interval payable in arrears for a number of intervals.
- **Series of Payments** - Knowing the rate of interest per interval and the number of intervals.

In implementing the above procedures it has often be necessary to find solutions of polynomial equations. In order to find these solutions we have used the following techniques:

- **Interval Bisection Method** - A robust method that always finds a solution or a singularity inside a bracketed interval.
- **Newton-Raphson Method** - Given a first approximation to a root and the differential of the function this procedure will always produce a solution. We implement this procedure for polynomial functions of one variable.

Interest Calculations

- **Exponents and Series** - Law of Exponents, Arithmetic Progression, Finite/Infinite Geometric Series
- **Simple interest** - a deposits value, Real worth, Real return
- **Compound interest** - Accumulated values, Real worth, Real return, Depreciation
- **Effective and nominal interest** - Real return, Force of interest
- **Accumulated values of annuity-certain** - Accumulated annuity certain in arrears, Accumulated annuity certain in advance
- **Present values**
- **Present value of annuity-certain**
- **Yield** - Internal rate, Real and nominal
- **Real returns** - Bonds, Rate of return

This product also contains the following features:

- **GUI Bundle** - we bundle a suite of graphical user interface JavaBean components (with 1, 2, 4 or site-wide license) allowing the developer to plug-in a wide range of GUI functionality (including charts/graphs) into their client applications
- **EAR Files** - we provide individual customized EAR files for the most widely used application servers including IBM WebSphere 4.0/5.0, BEA WebLogic 6.1/7.0, Oracle 9iAS, Sun ONE AppServer 7, Ironflare Orion 1.5.2/1.6.0, Borland AppServer 5.0, Sybase EAServer 3.6 and JBoss 2.4.4/3.0.0
- **Self-Deploy** - the relevant servers EAR file will be self-deployed onto supported local application servers during the installation of the self-install package. The supported application servers include IBM WebSphere 4.0/5.0, BEA WebLogic 6.1/7.0, Oracle 9iAS, Borland AppServer 5.0, Ironflare Orion 1.5.2/1.6.0 and JBoss 2.4.4/3.0.0
- **JDBC Mediator** - A server side EJB component which mediates between an EJB component, clients and DBMS. The mediator moves most of a clients JDBC calls to the server and hence greatly increases the speed of JDBC client applications.
- **Web Application Example** - A JSP interactive HTML interface which enables you to test every component method directly from your browser.
- **Synthetic JDBC** - we use a JSP component to perform EJB calculations on SQL database columns from a remote DBMS. We apply an EJB function to certain rows from the database and list the output in HTML format. This is a powerful feature since it allows you to perform calculations in a JDBC manner without having to code the EJB-to-JDBC transaction yourself as it is all done by the JSP within the Application Server.

- **UML Models** - to assist system architects we provide UML diagrams of this component

1.2 Package Details

The Bonds J2EE Application contains the following:

- 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
 - UML Models
 - Guide to WebCab Components
- JavaDocs Documentation
 - Class Descriptions
 - Methods Descriptions
- Deployment Archives (EAR)
- Examples and Related Source Code Files
- WebCab Components Brochure

1.3 Prerequisites and Compatibility

1.3.1 System Requirements

- An Operating System running Java™
- Pentium® III 733 MHz
- 256MB RAM
- A J2EE1.3 (EJB 2.0) compatible Application Server

1.3.2 Compatibility

Operating System for Deployment

- Windows 2003, XP, 2000, NT
- Sun Solaris
- Linux
- IBM AIX
- HP-UX

Component Type

- Enterprise JavaBeans™

Built Using

- Java™ 2 SDK Standard Edition 1.4.2
- Java™ 2 SDK Enterprise Edition 1.3

Compatible Application Servers

- IBM WebSphere Application Server V4.0/V5.0
- BEA WebLogic® Server 6.1/7.0
- Oracle® 9i Application Server
- Sun One Application Server 7
- Borland Enterprise AppServer 5.0
- Ironflare Orion Server 1.5.2/1.6.0
- Sybase EAServer 3.6
- JBoss 2.4.4/3.0.0

Chapter 2

Where do I Start?

Start using the Bonds J2EE Application right away by following the few quick and simple steps described in this chapter. We provide support for most Java compatible operating systems, such as Windows, Linux, IBM AIX, HP-UX and Solaris. Along with the most widely used Application Servers including IBM WebSphere, BEA WebLogic, Oracle9i, Sun ONE AppServer, Borland AppServer, Sybase EAServer, Ironflare Orion, and JBoss. If you require additional information regarding the use of this J2EE Application, then please don't hesitate to contact us via our support forum at: <http://www.webcabcomponents.com/support/index.php>.

2.1 Self-Deploy Installer

If you are locally using one of the following application servers:

- IBM WebSphere V4.0/V5.0
- BEA WebLogic 6.1/7.0
- Oracle9i v9.0.2/v9.0.3
- Borland Enterprise AppServer 5
- Ironflare Orion 1.5.2/1.6.0
- JBoss 2.4.4/3.0.0

During the installation of this Application, you were prompted to follow the deployment procedure of your application server. In this case the Application should already be deployed on your application server and you can start using it immediately.

If the Application is not installed on your local application server or you wish to deploy on a remote application server then you should follow the relevant steps below.

2.2 Deploying this Component

Depending on the Enterprise platform you have chosen for distributing your business components, you will follow different procedures when starting off. Within this chapter we describe individual Application Server deployment procedures. Please feel free to skip to the section that refers to the Application Server you are planning to use.

In case your Application Server is not listed in this chapter or the installation of your Application fails, please send us an email to Support@WebCabComponents.com attaching any logs or error messages.

2.3 IBM WebSphere Application Server

The deployment procedures for IBM WebSphere V4.0 and V5.0 are the same. The only difference is that on WebSphere V4.0 you will be deploying the EJB1.1 version of Bonds, whereas on WebSphere V5.0 you will be able to install the EJB2.0 version as well.

2.3.1 Deploying on IBM WebSphere V4.0 and V5.0

Starting the Server

The WebSphere Application Server (WAS) requires to be up and running at deployment time. Start the server under Windows by selecting **Programs→IBM WebSphere→Application Server V4.0** (resp. **V5.0**). Alternatively, the server can be started by running the `startServer.bat` script from the installation path. Under Linux log in as `root`, change to the `${WAS_HOME}/bin` directory, where `${WAS_HOME}` is the WebSphere installation path, and type the following line at command prompt¹:

```
./startServer.sh
```

Connecting to the Administration Console

Connect to the WAS Administrative Console, by opening a browser and typing in the following URL address:

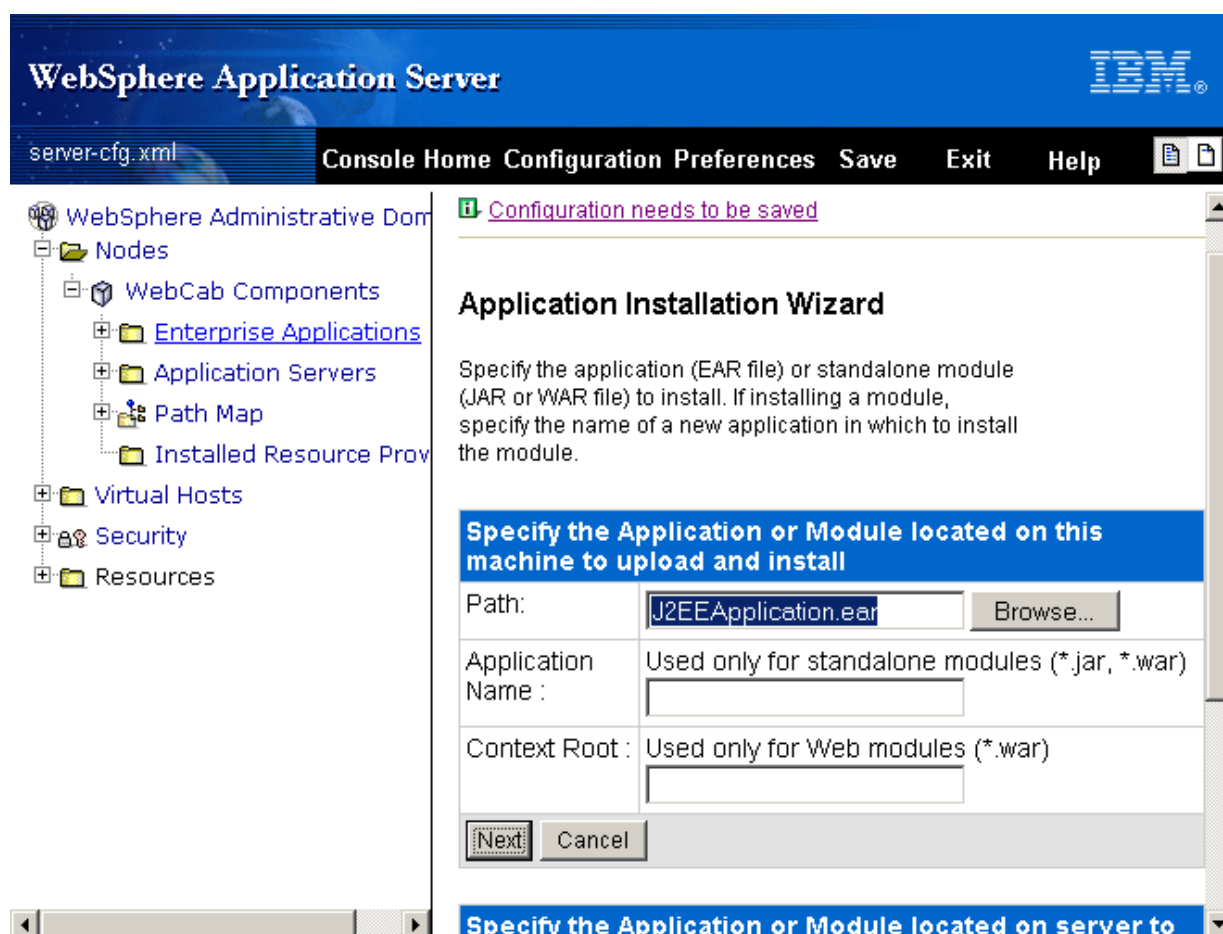
`http://localhost:9090/admin`

where `localhost` can be replaced by the hostname of the computer running WebSphere when connecting from a client machine. The browser will bring up a login page and ask you to fill in a name in order to track user-specific changes to configuration data. You may leave the field blank and press the **Submit** button. If an Alert page is displayed, press the **Cancel** button and continue.

¹ `${WAS_HOME}` usually defaults to `/opt/WebSphere/AppServer` under Unix.

Deployment

The Administrative console manages the Application Server into nodes and each node runs its own Enterprise Applications. You will be deploying inside one of the existing nodes which you can select from the left panel by expanding the **Nodes** folder. Expand the name of the installation node and select the **Enterprise Applications** item. The browser will bring up in the central frame a list of all currently installed Enterprise Applications. Click the **Install** button, browse the **Bonds.ear** file, as installed under the *[ROOT_DIR]/Bonds/Deployment/IBM WebSphere V4.x²* directory and click **Next³**.



IBM WebSphere V4.0/V5.0 Application Installation Wizard Page.

Click again **Next** through every screen until you reach the **Mapping Resource References to JNDI Names** screen. You will be required to map certain resource references to WebSphere DataSource JNDI names⁴. Keep clicking **Next** in every screen and press **Finish** to complete the deployment.

²If this is the V5.0 version of WebSphere, consider browsing from *[ROOT_DIR]/Bonds/Deployment/Ejb2.0/WebSphere V5.0*.

³*[ROOT_DIR]* represents the Bonds installation directory.

⁴Read the IBM WebSphere documentation for information about how to define DataSources.

After waiting for a few minutes, the Administrative Console will display a page where Bonds will be listed among the currently installed Enterprise Applications.

Restarting the server

When installing Bonds for the first time inside WebSphere, you will be required to restart the IBM server before running it. You will also need to save the current configuration before doing so. Click on the *Configuration needs to be saved* link located at the top of the page and confirm the save by pressing OK.

In order to stop the server, click the **Application Servers** folder from the left pane under the current node and press the **Stop** button. In the next page click **OK** and wait for the console to confirm a successful shutdown. The next time you start the WebSphere server the Bonds Application will be installed and made available to all clients.

2.4 BEA WebLogic Server

2.4.1 BEA WebLogic Server 6.1

Enterprise JavaBean™ components and J2EE Application are deployed inside a running WebLogic 6.1 server through a web browser interface. On this server you will be installing the EJB1.1 version of Bonds.

Starting the server

Start BEA WebLogic 6.1 under Windows from the **Programs→BEA WebLogic E-Business Platform→WebLogic Server 6.1→Start Default Server** icon in the Start Menu. If you're running WebLogic 6.1 under Linux, change the current directory to `/home/user/boa/wlserver6.1/config/mydomain`, where *user* is your Unix account and *mydomain* points to the default domain name. Within this folder type the following line at command prompt:

```
./startWebLogic.sh
```

If asked for, enter the login password and wait a few seconds until the server to initializes.

Connecting to the WebLogic Console

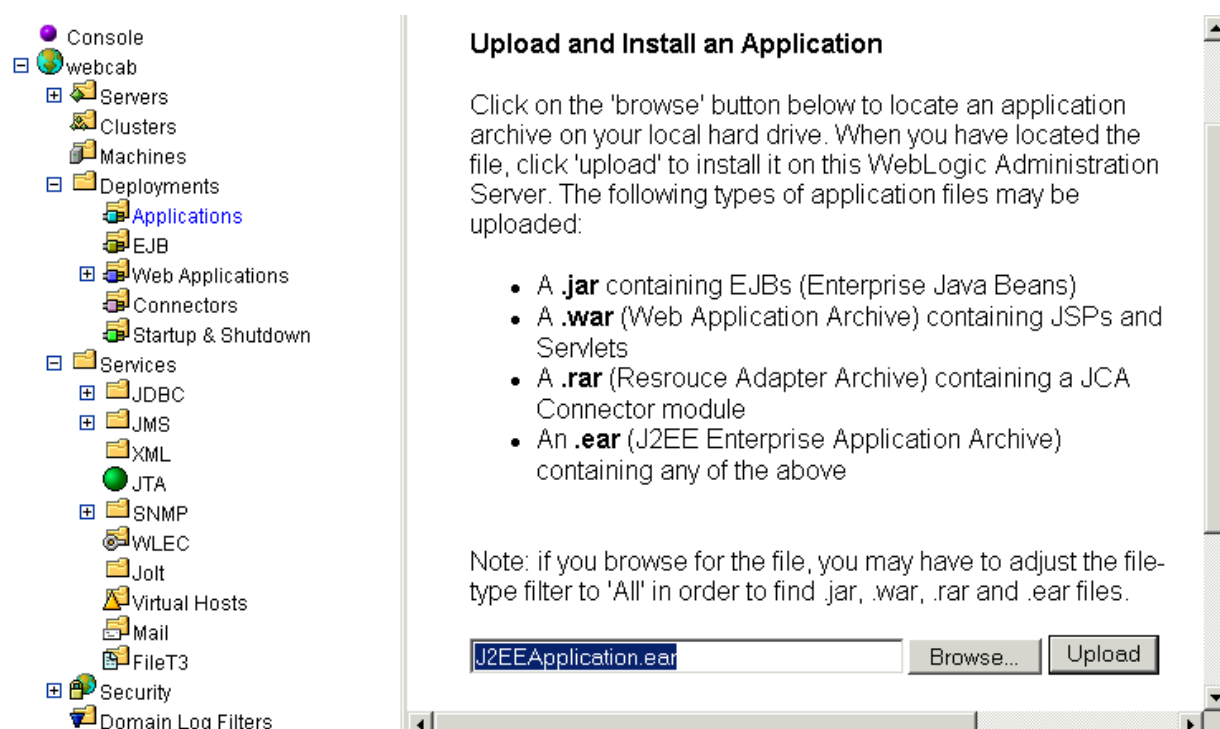
When the WebLogic Server is up and running, open an Internet browser and type in the following address:

`http://localhost:7001/console`

where `localhost` can be replaced by the hostname of your server running WebLogic, when connecting from another client machine. At the login screen enter the user name ("system" by default) and type in the login password. The browser will bring up the BEA WebLogic Server support interface.

Deployment

You will see in the left side of the screen your domain's name and a list of currently deployed Enterprise Applications. Click on the **Applications** item under the **Deployments** folder, then click the *Install a new Application...* link in the right panel.



WebLogic Server 6.1 Upload and Install Screen.

Browse the `Bonds.ear` archive from `[ROOT_DIR]/Bonds/Deployment/BEA WebLogic 6.1` directory and click the **Upload** button⁵. The J2EE Application will be uploaded and deployed inside the WebLogic Server 6.1.

2.4.2 BEA WebLogic Server 7.0 (J2EE 1.3 compliant)

Enterprise JavaBean™ components are deployed inside a running WebLogic 7.0 server through a web browser interface. Since WebLogic Server 7.0 is J2EE1.3 compatible, you will be installing the EJB2.0 version of Bonds.

Starting the server

Start BEA WebLogic 7.0 under Windows from the `Programs→BEA WebLogic Platform 7.0→User Projects→mydomain` folder in the Start Menu, by clicking the “Start WebLogic Server” icon. Alternatively, you may start the server by running the `startWLS.cmd` script inside the `C:\bea\weblogic700\server\bin` folder.

⁵`[ROOT_DIR]` is the Bonds installation directory, as specified at installation time.

If you're running WebLogic 7.0 under a Linux platform, change the current directory to `[WEBLOGIC_HOME]/weblogic700/server/bin`, where `[WEBLOGIC_HOME]` points to the BEA home directory, as installed under Linux, for example `/home/user/boa`. In this folder type the following line at command prompt:

```
./startWLS.sh
```

If asked for, enter the login password and wait a few seconds until the server initializes.

Connecting to the WebLogic Server Console

When the BEA WebLogic Server is up and running, open an Internet browser and type in the following address:

```
http://localhost:7001/console
```

where `localhost` can be replaced by the hostname of the server running WebLogic, when connecting from another client machine. In the login field, enter your user name ("system" by default) and type in the login password. The browser will bring up the BEA WebLogic Server 7.0 Console.

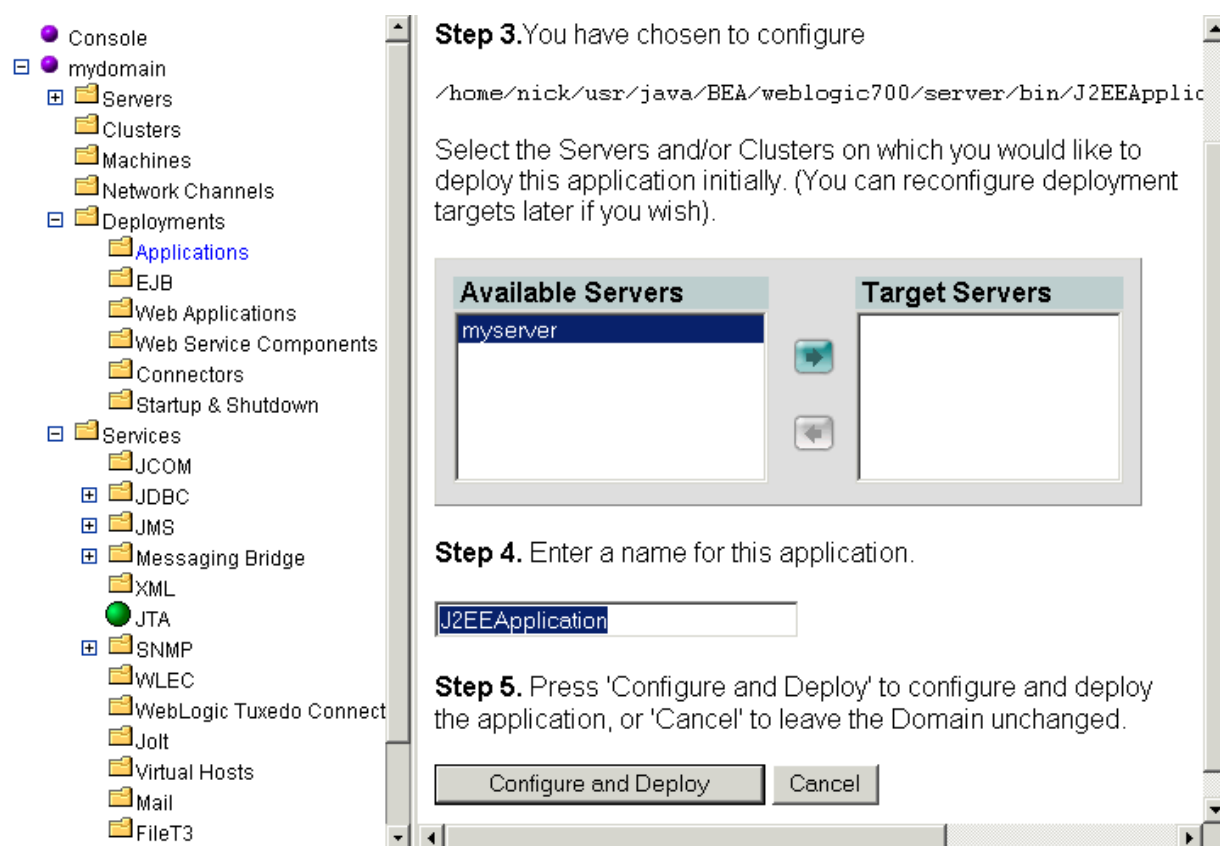
Uploading the Application

You will see in the left side of the screen your domain name and a list of currently deployed Enterprise Applications under the **Deployments/Applications** tree. Click on the **Applications** item under the **Deployments** folder, and press *Configure a new Application...* in the right panel. Click the link that says "upload it through your browser" and browse the `Bonds.ear` archive from `[ROOT_DIR]/Bonds/Deployment Ejb2.0/WebLogic 7.0` directory and click the Upload button⁶.

Deployment

After the upload is complete the page will display at the bottom of the screen the name of the uploaded file, `Bonds.ear`. Click the "[select]" link at its left.

⁶`[ROOT_DIR]` is the Bonds installation directory, as specified at installation time.



WebLogic Server 7.0 Deployment Page.

In the **Step 3.** section choose an available server and add it to the Target Servers box by pressing the right arrow. As a final step, press the **Configure and Deploy** button in the **Step 5.** section.

The next page will guarantee a successful deployment if the Status at the bottom of the screen displays a message that says “Running”.

2.5 Oracle9i Application Server

We support installation under Oracle9i v9.0.2 and v9.0.3 Application Servers. The deployment procedures for Oracle9i 9.0.2 and 9.0.3 are quite similar. The major difference is that on Oracle9i 9.0.2 you will be deploying the EJB1.1 version of Bonds, whereas on Oracle9i 9.0.3 you will be able to install the EJB2.0 version as well.

2.5.1 Oracle9i v9.0.2

In order to deploy the EJB1.1 version of Bonds on your Oracle9i 9.0.2 Application Server, follow these three simple steps:

1. **Change current directory to the Oracle9i Application Server installation path.** Usually it is located in the *j2ee/home* subfolder of your Oracle9i DBMS home

directory. You can identify this folder by looking for a file named *oc4j.jar* (Oracle Containers for J2EE).

2. **Start Oracle9i v9.0.2.** In a command prompt window, type the following line:

```
java -jar oc4j.jar
```

3. **Deploy Bonds.** In another command prompt window, type the following two lines:

Line 1

```
java -jar admin.jar ormi://localhost admin "password" -deploy -file  
"[ROOT_DIR]/Bonds/Deployment/Oracle9i/Bonds.ear"  
-deploymentName "Bonds"
```

Line 2

```
java -jar admin.jar ormi://localhost admin "password"  
-bindWebApp Bonds Bonds http-web-site Bonds
```

Note that *password* is your administration password and *[ROOT_DIR]* is the installation directory of Bonds.

2.5.2 Oracle9i v9.0.3 (J2EE1.3 certified)

In order to deploy the EJB2.0 version of Bonds on your Oracle9i 9.0.3 Application Server, follow these three simple steps:

1. **Change current directory to the Oracle9i Application Server installation path.** Usually it is located in the *j2ee/home* subfolder of your Oracle9i DBMS home directory. You can identify this folder by looking for a file named *oc4j.jar* (Oracle Containers for J2EE).
2. **Start Oracle9i v9.0.3.** In a command prompt window, type the following line:

```
java -jar oc4j.jar
```

3. **Deploy Bonds.** In another command prompt window, type the following two lines:

Line 1

```
java -jar admin.jar ormi://localhost admin "password" -deploy -file  
"[ROOT_DIR]/Bonds/Deployment Ejb2.0/Oracle9i v9.0.3/Bonds.ear"  
-deploymentName "Bonds"
```

Line 2

```
java -jar admin.jar ormi://localhost admin "password"
```

```
-bindWebApp Bonds Bonds http-web-site Bonds
```

Note that *password* is your administration password and *[ROOT_DIR]* is the installation directory of Bonds.

2.6 Sun ONE AppServer 7

Starting the Application Server

On Windows, click the Windows Start button, then choose **Programs→Sun ONE Application Server 7→Start Application Server**. If you are working under Unix, change the current directory to *\$SUN_HOME/bin*, where *\$SUN_HOME* is the Sun ONE installation path and type the following line at command prompt:

```
asadmin start-appserv
```

Starting the Administration Console

After the Sun ONE server initializes, open an Internet browser and type in the following address:

```
http://localhost:4848/admin
```

where *localhost* can be replaced by the hostname of your server running Sun ONE, if you are connecting from a different machine. At the login screen enter the administrator user name (“admin” by default) and type in the login password. The browser will bring up the Sun ONE Administration Console.

Deployment

Choose **Applications→Enterprise Apps** in the left-hand panel and click the **Deploy...** button in the page on the right. Browse the *[ROOT_DIR]/Bonds/Deployment Ejb2.0/Sun ONE/Bonds.ear* file where *[ROOT_DIR]* is the Bonds installation directory and click OK. Press again OK in the next screen and allow a couple of minutes to complete the deployment process.

2.7 Borland Enterprise AppServer 5

In order to deploy this Application inside Borland AppServer 5.0, you will need to perform the following tasks in the following order:

1. **Start the Borland Enterprise Server**

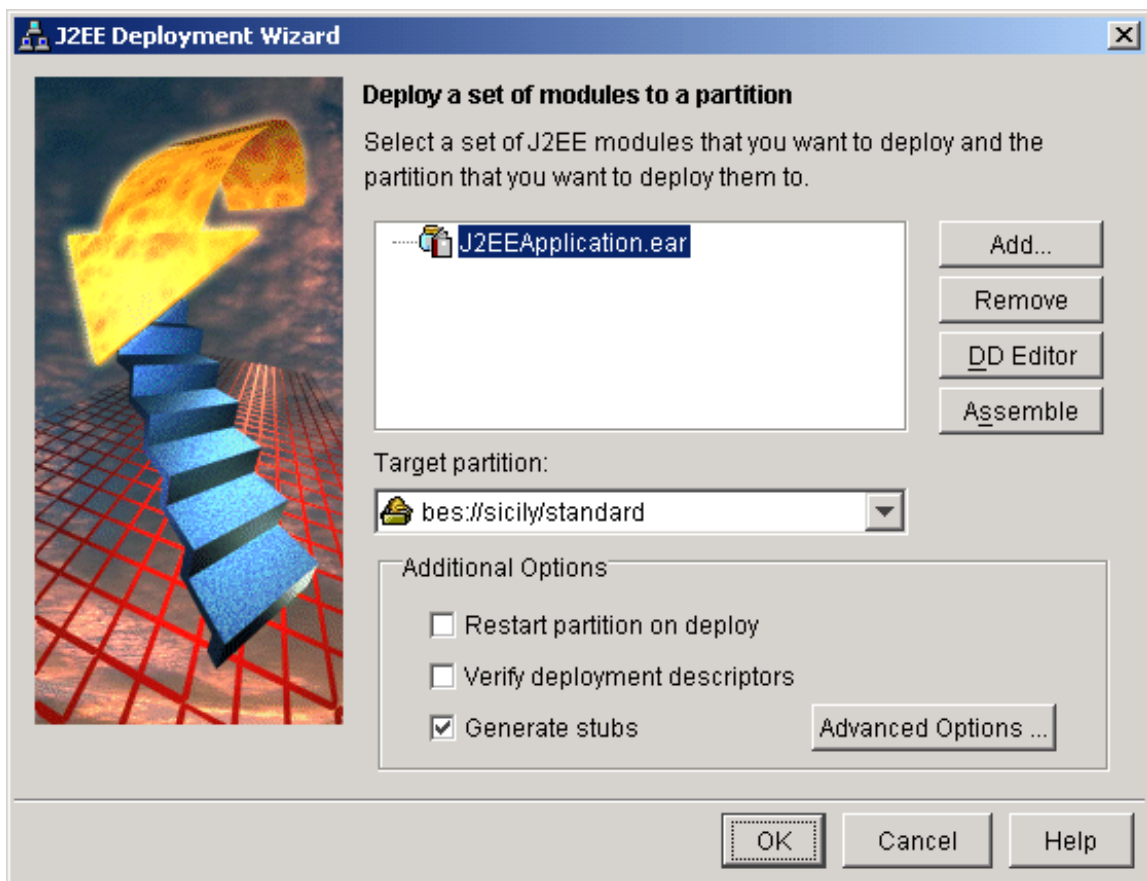
Under Windows you will click the **Borland Enterprise Server→Server** icon from the Start menu and wait a couple of seconds for the server to initialize. Under Unix type: *bin/ias* at the command prompt from the installation directory.

2. Open the Administration Console

You can open the Borland Enterprise Administration Console by clicking the **Borland Enterprise Server**→**Console** icon from the Start menu, if you are running under the Windows operating system. A login dialog will pop up, where you can click the **Login** button without making any modifications at all. Under Unix type: `bin/console` from the installation directory.

3. Deploy Your J2EE Modules

Select **Tasks**→**Deployment**→**Deploy J2EE modules to a partition...** to open the J2EE Deployment Wizard. Click the **Add** button and select the **Bonds.ear** archive from the `[ROOT_DIR]/Bonds/Deployment Ejb2.0/Borland AppServer 5` folder, where `[ROOT_DIR]` is the root directory where Bonds was installed.



Borland AppServer 5 Deployment Dialog.

At the bottom of this dialog, turn on the **Generate Stubs** option and click **Next**. The deployment process will scroll inside a text area. When it completes, press the **OK** button.

2.8 Sybase EAServer 3.6

Deployment inside EAServer 3.6 is performed using the Jaguar Manager inside a running Jaguar Server.

1. **Start you Jaguar Server**

Under Windows, open Explorer, navigate to the EAServer installation folder and run the `serverstart_jdk12.bat` file by double-clicking it. If you are running the Jaguar Server under Unix, change your current directory to the EAServer installation path and type in the following line at command prompt:

```
./srvstart_jdk12
```

2. **Deploy to your Jaguar Repository**

Open the Jaguar Manager and connect to the previously started Jaguar server. Right-click with your mouse the **Packages** folder under **Sybase Central Java Edition**→**Jaguar Manager** and select **Deploy**→**EJB 1.1 Jar** from the popup menu. In the **Deploy Wizard** dialog, browse the `[ROOT_DIR]/Bonds/Deployment/Sybase/Bonds.ear` file⁷ and click **Next**. Wait for a few seconds in order to allow the deployment to take place and press the **Close** button when done.

3. **Deploy Inside your Jaguar Server**

From the Jaguar Manager, right-click the **Installed Packages** folder from the left panel and select **Install Package...** in order to bring up the **Package Wizard** dialog. Click on the **Install an Existing Package** button and select **Bonds** from the list. Press **OK** to complete deployment inside EAServer.

2.9 Ironflare Orion Server

We support installation under Ironflare Orion 1.5.2, 1.5.4 and 1.6 Servers. The deployment procedures for Orion 1.5.2 and 1.6 are quite similar. The major difference is that on Orion 1.5.2. you will be deploying the EJB1.1 version of Bonds, whereas on Orion 1.6 you will be able to install the EJB2.0 version as well.

2.9.1 Orion 1.5.2 (and Orion 1.5.4)

In order to deploy the EJB1.1 version of Bonds on your Orion 1.5.2 Server, follow these three simple steps:

1. **Change current directory to the Orion Server installation directory.** You can identify it knowing it contains a file named *orion.jar*.
2. **Start Orion Server 1.5.2** In a command prompt window, type the following line:

⁷`[ROOT_DIR]` is the root directory where Bonds was installed.

```
java -jar orion.jar
```

3. **Deploy Bonds.** In another command prompt window, type the following two lines:

Line 1

```
java -jar admin.jar ormi://localhost admin "password" -deploy -file  
"[ROOT_DIR]/Bonds/Deployment/Ironflare Orion 1.5.x/Bonds.ear"  
-deploymentName "Bonds"
```

Line 2

```
java -jar admin.jar ormi://localhost admin "password"  
-bindWebApp Bonds Bonds default-web-site Bonds
```

Note that *password* is your administration password and *[ROOT_DIR]* is the installation directory of Bonds.

2.9.2 Orion 1.6 (EJB2.0 compliant)

In order to deploy the EJB2.0 version of Bonds on your Orion 1.6 Server, follow these three simple steps:

1. **Change current directory to the Orion 1.6 Server installation path.** You can identify this folder by looking for a file named *orion.jar*.
2. **Start Orion Server 1.6** In a command prompt window, type the following line:

```
java -jar orion.jar
```

3. **Deploy Bonds.** In another command prompt window, type the following two lines:

Line 1

```
java -jar admin.jar ormi://localhost admin "password" -deploy -file  
"[ROOT_DIR]/Bonds/Deployment Ejb2.0/Orion 1.6/Bonds.ear"  
-deploymentName "Bonds"
```

Line 2

```
java -jar admin.jar ormi://localhost admin "password"  
-bindWebApp Bonds Bonds default-web-site Bonds
```

Note that *password* is your administration password and *[ROOT_DIR]* is the installation directory of Bonds.

2.10 JBoss Open Source Application Server

We provide deployment instructions and resources for JBoss 2.4.4 (with TomCat 4.0.1) and 3.0.0 (with TomCat 4.0.3). Under JBoss 3.0.0 or later you will be deploying the EJB2.0 version of Bonds. All previous JBoss versions will only support the EJB1.1 Application.

2.10.1 JBoss 2.4.4 (and JBoss 2.4.3)

Start the JBoss Server by changing the current directory to *[JBOSS_HOME]/jboss/bin*, where *[JBOSS_HOME]* is the installation path for your JBoss 2.4.4 release. Under Windows, type at command prompt:

```
run.bat
```

If you are running JBoss under Linux, you will type in the following:

```
./run.sh
```

and wait for a few seconds until the JBoss server is up and running.

In order to deploy Bonds, you will need to make a copy of the **Bonds.ear** Enterprise archive located inside the *[ROOT_DIR]/Bonds/Deployment/JBoss* directory into the *[JBOSS_HOME]/jboss/deploy* folder⁸.

2.10.2 JBoss 3.0.0 (EJB2.0 compliant)

Start the JBoss Server by changing the current directory to *[JBOSS_HOME]/bin*, where *[JBOSS_HOME]* is the installation path for your JBoss 3.0.0 release. Under Windows, type at command prompt:

```
run.bat
```

If you are running JBoss 3.0.0 under Linux, you will type in the following:

```
./run.sh
```

and wait for a few seconds until the JBoss server is up and running.

In order to deploy Bonds, you will need to make a copy of the **Bonds.ear** Enterprise archive located inside the *[ROOT_DIR]/Bonds/Deployment Ejb2.0/JBoss 3.0.0* directory into the *[JBOSS_HOME]/server/default/deploy* folder⁹.

⁸*[ROOT_DIR]* is the root directory where you installed Bonds.

⁹*[ROOT_DIR]* is the root directory where you installed Bonds.

2.11 Testing the Component

2.11.1 Accessing an Online Demo

By far the easiest way to test the functionality of this J2EE Application is to view the online demo. The online demo can be accessed from our [J2EE Homepage](#) by clicking the ‘[Demo]’ link corresponding to this Application.

2.11.2 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.

2.11.3 Selecting a method to test

After clicking on the ‘[Demo]’ link for this application from our home page the J2EE Web demo will launch within your web browser. To order to select a method from this application’s J2EE 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 J2EE 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 J2EE component menu and left click a function to select it.

2.11.4 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.

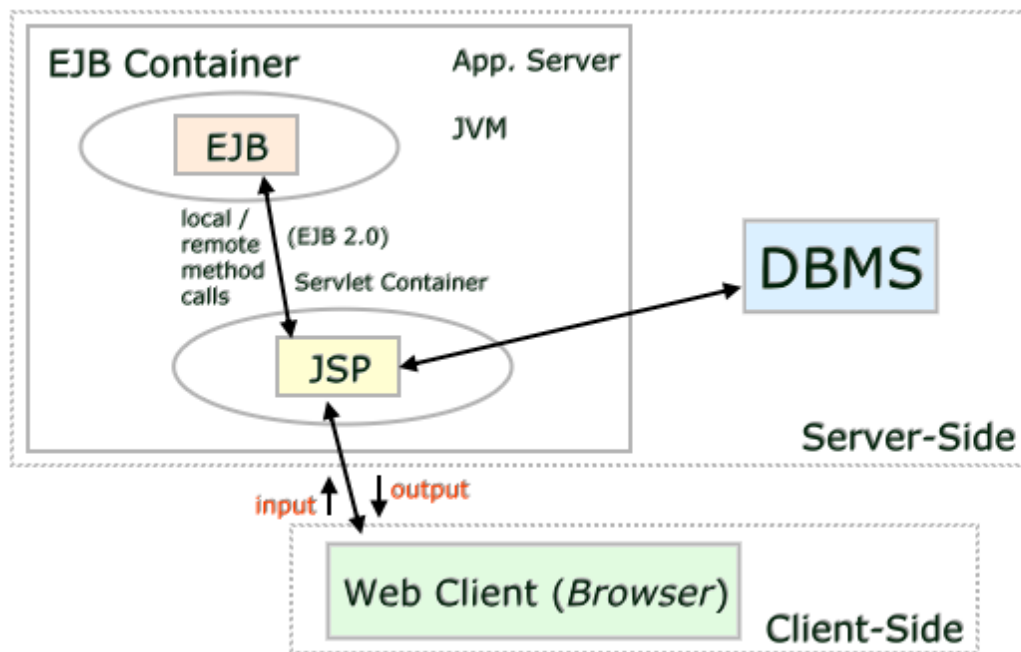
Running the demo using text boxes

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 J2EE 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:



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

2.12 Using the Web Demo

Within the EAR for this component, we have also included a Web Application Demo for this component. If you have deployed the EAR file on your Application Server, then this JSP Web Application demo should be installed within your Servlet container.

In order to run this Web Demo, please start an Internet browser and type in the corresponding address (listed below) for your application server, or simply click the link, which corresponds to your Application Server:

- IBM WebSphere - <http://localhost:9080/BondsWebExample>
- BEA WebLogic - <http://localhost:7001/BondsWebExample>
- Oracle 9iAS - <http://localhost:8888/BondsWebExample>
- Sun ONE Application Server - <http://localhost:81/BondsWebExample>
- Borland AppServer - <http://localhost:8080/BondsWebExample>
- Ironflare Orion - <http://localhost:8888/BondsWebExample>
- JBoss - <http://localhost:8080/BondsWebExample>

Remark This J2EE Web Application Example has also been deployed on our server. You can access it online by [clicking here](#).

Chapter 3

Fundamental Theory of Bonds

Within this chapter we detail the fundamental parts of the theory of bonds including zero rates, pricing, forward rates, duration, simple hedging strategies and convexity.

3.1 Types of Rates and pricing

3.1.1 Types of Rates

As we have seen in the previous chapter the yield of an interest bearing investment can be quoted in a number of ways. Moreover, each type of interest based product for example mortgages, deposits, loans will each have their own conventions and each of the local markets on which these products are traded will also introduce another layer of conventions. Within the bond market for example there are a number of ways in which the ‘yield’ of a bond may be quoted. The particular convention which is used will depend on the type of bond and the market in which it is traded.

For interest rate derivative products the three following examples are of particular importance:

- **Treasury Rates** - this is the rate at which a government can borrow money in its local currency
- **LIBOR Rates** - LIBOR stands for ‘London Interbank Offer Rate’ and is the rate at which one large international bank will lend money to another large international bank. Generally, LIBOR is quoted as the ‘overnight rate’ in US dollars.
- **Repo Rate** - the implied rate from a repurchase agreement. Securities are sold for the exchange of cash and repurchased at a prior agreed rate. The Repo rate is negotiated between the two parties.
- **n -year zero rate** - the rate of interest earned on an investment over n years where the principle and interest payments are only realized after n years.

3.1.2 Bond Pricing

The price of a government bond (i.e. assuming no default risk) is determined by discounting the future cash flows. A corresponding corporate bond (i.e. with default risk) will have a lower market price since investors will demand a yield premium to make allowance for the default risk.

Implemented Methods

Within our EJB Component we offer following general bond pricing functionality:

- `TreasuryPrice.priceOfBond` - A general method which allows a government backed bond to be priced by discounting according to the continuously compounded zero rates.
- `TreasuryPrice.tbondPrice` - A general method which allows a government backed bond to be priced by discounting according to the risk free interest rate.
- `TreasuryPrice.zeroTBondPrice` - Evaluates the price of a zero Treasury bond is discounting according to the risk free interest rate.

We also offer methods by which the yield to maturity. In each case we deduce to yield to maturity from the market price of the bond and the coupon payments until maturity. In particular, we offer:

- `TreasuryPrice.yieldTomaturity` - Calculates the yield to maturity (YTM) (also referred to as the internal rate of return (IRR)) for a general bond investment.
- `TreasuryPrice.zeroYieldToMaturity` - Calculates the zero to maturity of the zero coupon bond.
- `TreasuryPrice.yieldToMaturityFromPrice` - The yield to maturity is derived from the price of the bond and future payments of coupons and principle sum.

Remarks

1. Note that the yield to maturity for non-government backed bonds will be greater than the risk free rate in the relevant currency because of the bonds default risk will need to be reflected within its price. Therefore, the price and yield to maturity for the Treasury bond will act as a lower respectively upper bound on the price and yield to maturity of a similar non-government backed bonds (i.e. a corporate bond).
2. In order to evaluate the yield to maturity we where required to solve polynomial equations of one variable relating the yield, price and cash flows. In each case we solved these polynomial expression by applying the Newton-Raphson method.

Finally, a method for evaluating the Par yield of a Treasury bond is provided:

- `TreasuryPrice.parYield` - The Par yield is evaluated for a Treasurty bond where the number of yearly payments and the abbuity can very.

3.1.3 Zero Rates, Bootstrapping and the Constructing the Zero Rate Curve

We offer functionality which allows the zero rates to be deduced from zero coupon bonds and then by an iterative procedure known as bootstrapping from coupon paying bonds. Once a set of zero rates for various maturities is known we are able to construct via the linear interpolation approach the zero coupon curve for all maturities. We offer this functionality with the following methods:

- `CalculatingZeroRates.zeroRateFromZeroBond` - Evaluation of the zero rate for a given maturity from the corresponding zero bond.
- `CalculatingZeroRates.zeroRateBootStrap` - Bootstrap method allows the zero rate for the maturity of a coupon paying bond to be deduced by an iterative procedure from coupon paying bonds when the zero rates of the maturities equal to the time until the coupons are paid are known.
- `CalculatingZeroRates.zeroRateCurve` - Construction of the zero rate curve from a finite set of zero rates of differing maturities in accordance to the convention that the zero rate curve is linear between the given zero rates and constant outside the range of maturities.

Application of the Zero Curve

There will be instances when you will require to know the zero rate for a maturity which cannot be deduced from traded zero coupon and coupon paying bonds. However within most interest rate markets there is the range of zero and coupon paying bonds which will allow the deduction of the exact zero rates for a range of maturities.

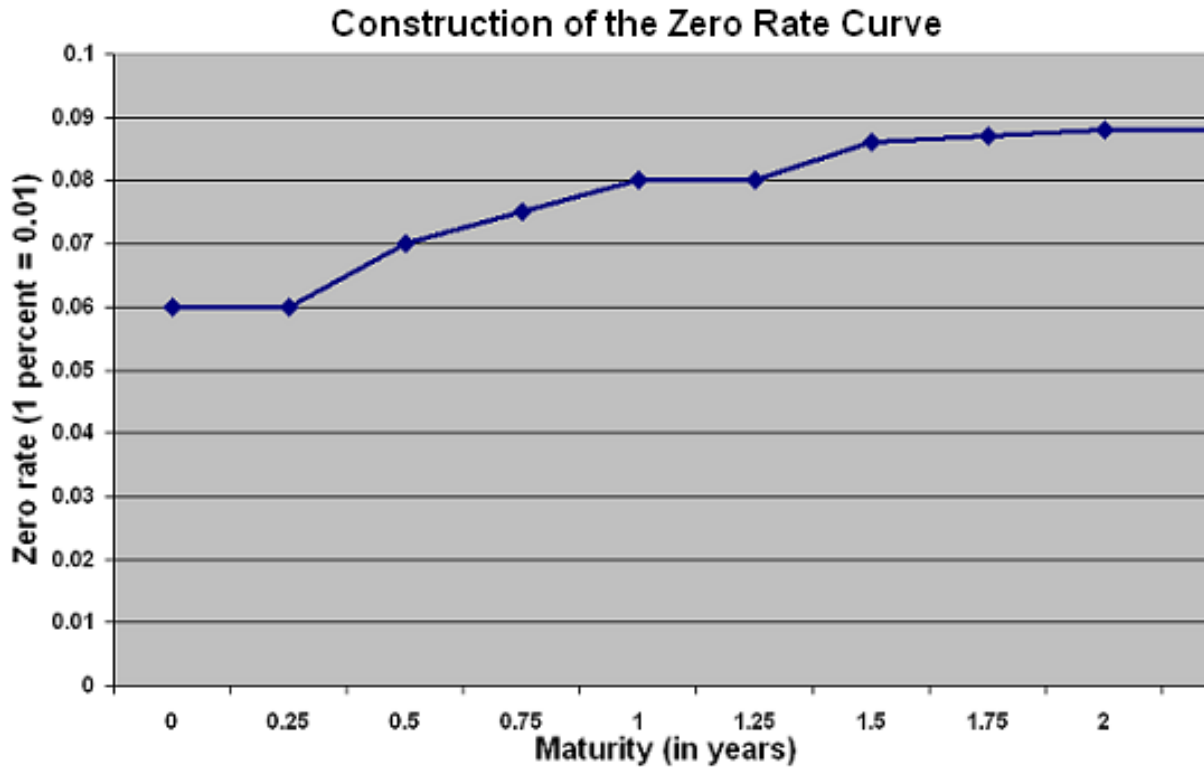
To solve this problem we can use these maturity to construct what is known as the 'zero curve', by interpolating a curve around the finite number of known zero rates which have been explicitly calculated. It is customary for the zero curve to be constructed using linear interpolation between the known zero rates. It is also the custom that the constructed zero curve is assumed to be 'flat' (or constant) between the zero maturity and the lowest maturity explicitly calculated, and above the highest maturity.

Example

If we know the following set of (present) zero rates of US Treasury bonds:

Maturity (in years)	0.25	0.5	0.75	1	1.25	1.5	1.75	2
Zero rate (in %)	6	7	7.5	8	8	8.6	8.7	8.8

then by using the linear interpolation technique for the construction of the zero rate curve we have:



Once we have constructed the zero rate curve we can read off values for all intermediate values.

Remark The above constructed zero rate curve is constant between maturities of 0 and 0.25 years, and for maturities larger than 2 years.

3.2 Forward Rates and Forward-Rate Agreements

3.2.1 Evaluating the Forward Rate

The forward rate for a given period (and compounding convention) is the interest rate which is implied from the current zero rates. To illustrate forward rates we will consider two straightforward examples, one where the interest is compounded continuously (Example 1) and the other where it is compounded over two discrete periods (Example 2).

Example 1: We wish to find the n th year annual forward rate where the interest is compounded continuously. The $(n - 1)$ th and n th year zero rates is known to be $a\%$ and $b\%$. Now since total payments due from the continuously compounded interest of $a\%$ per annum on a principle sum of $\$M$ after one year is:

$$(\exp a)M$$

and hence the payment due after the second year (when we will earn a rate of $b\%$) is going to be:

$$\exp b((\exp a)M) = (\exp a + b)M = \left(\exp \left(\frac{a+b}{2} \right) 2 \right) M$$

Or put another way, the forward rate (per annum) over the period is $\frac{a+b}{2}$, or the arithmetic average of the two rates.

Example 2: Say now that in the above example the interest is compounded once per annum. Now the interest payments after one and two years is $((100 + a)M)/100$, and $((100 + b)(100 + a)M)/1000$. Which leads us to the following equation:

$$\frac{(100 + b)(100 + a)M}{1000} = \frac{(100 + c)^2 M}{1000}$$

where c is the zero rate (in accordance with the definition). Simplifying this expression gives us a value for c of:

$$c = \frac{\sqrt{(100 + a)(100 + b)}}{100}$$

This is precisely the forward rate, that is, the unique (expected) rate at a future date which is implied by the current zero rates.

Remark The continuously compounded case will always imply a higher forward rate than the discretely compounded case. Since the interest payments will be compounded “much more often”. Saying this, the continuously compounded case and discretely compounded case often do not differ by a large margin. For example if $a = 5\%$, and $b = 6\%$ in the above examples in the continuous case the two year forward rate is 5.5% , whereas the annually compounded case the forward rate is 5.4988% .

Implementation

We have implemented within the **forwardRate** method within the **ForwardRates** EJB Component a general method which allows the forward rate to be evaluated for any period between two points where the corresponding zero rates are known.

3.2.2 Forward-Rate Agreements

A Forward-Rate Agreement (FRA) is an agreement between two parties to pay an agreed level of interest on a given principle sum within a given period. Generally speaking, when the FRA is set up for a given period the forward rate and the agreed rate are usually the same. Since an institution can obtain the forward rate on a sum without cost the initial FRA does not have any value and hence no cash will change hands at its instigation. Moreover, it is usually the case that the FRA is settled in cash at the beginning of the interest period. By the present value of the cash flows due from the agreement being evaluated the corresponding payment exchanged between the parties.

Implementation

Within our component we have implemented the **forwardRateAgreement** method within the **ForwardRates** :services: which evaluates the value of the FRA to the holder. We also offer a **settlement** method of the **ForwardRates** EJB Component which calculates the settlement of the FRA in the beginning of the interest rate period.

3.3 Duration

3.3.1 Price and Duration of a Bond

The duration can be thought of as the weighted average of the times when payments are made. Formerally, say a bond provides the holder with payments p_i at time t_i ($1 \leq i \leq n$). The bonds price, B , and its continuously compounded yield, y , are related by:

$$B = \sum_{i=1}^n p_i \exp(-yt_i)$$

The duration, D , of the bond is defined as:

$$D = \frac{\sum_{i=1}^n t_i p_i \exp(-yt_i)}{B}$$

Remark For a zero-coupon bond that matures in n years the duration is n .

The reason why the duration plays an important role in the hedging and risk analysis of interest rate instruments is because of the following relation:

$$\frac{\Delta B}{B} = -D\Delta y$$

where Δy represents a small parallel shift in the interest rate curve and ΔB represents the corresponding change in the price of the bond. That is, the percentage change in the bond price is equal to the duration multiplied by the size of the parallel shift in the yield curve.

3.3.2 Duration of a Bond Portfolio

The natural way to define the duration of a portfolio of bonds is as the weighted average of the duration of the individual bonds in the portfolio with the weights being proportional to the bond prices. Though each individual bond may depend on differing yield curves if we assume that each of these curves experiences a parallel shift Δy , then the change in the price of the bond portfolio is given by the product of the duration of the portfolio and the size of the parallel shift.

3.3.3 Rescaling the Duration

Naturally by rescaling the notion of duration according to different interest compounding conventions. We are able to evaluate the duration when the yield is not expressed as continuously compounded interest rates but as the interest rate with respect to another compounding convention (e.g. semi-annual). We provide methods which allow the duration and the associated change in the portfolio to be calculated in such instances.

3.3.4 Duration in a general setting

It is important to note that the notion of duration can be used for assets other than bonds. In fact any instrument which depends on interest rates has a corresponding notion of duration with respect to the interest rate it depends on. The interest rate risk at least for small parallel shifts of the relevant interest rate curve can be calculated by using the corresponding notion of duration. Further, as we will see duration based hedging techniques can also be applied to such situations.

3.3.5 Duration Based Hedging and Convexity

Duration Based Hedging

As we have seen the duration relates the change in the yield of an interest based investment with the change in the price of the same instrument. By applying the duration measure we can deduce the number of interest rate futures contracts required to hedge an interest rate investment for small parallel shifts.

Duration-Hedge Ratio

We consider the duration-hedge ratio (also referred to as the price sensitivity hedge ratio) of an interest rate dependent asset such as a bond portfolio or a money market security. The hedge ratio calculates the number of relevant interest rate futures contracts which need to be brought (or sold) in order to hedge against price changes resulting from small parallel shifts in the interest rate curve.

Problems with Duration based hedging

The notation of duration provides a simple approach to the hedging of interest rate based assets. However, the hedge produced is often far from perfect since the duration measure only takes into account the 'linear' part of the pricing function and further assumes that the shifts in the interest rate curve will always be parallel. In an attempt to deal with the 'non-linear' part of the pricing function the notion of convexity will be introduced. Non-parallel shifts of the interest rate curve are usually handled by breaking the curve into different time segments and hedging the interest rate risk on each of these segments. Such an approach is sometimes referred to as GAP management.

Remark The use of the hedge ratio for bonds is analogous to ‘delta hedging’ for equities. That is, it hedges the linear (or parallel) risk associated with the interest rate curve and does not take into effect the non-linear dependency of the assets on the underlying interest rate(s). Hence, in practice the hedge generated from the hedge ratio will need to be continually monitored and rebalanced accordingly.

Another issue which effects the performance of duration based hedging is when the contract can be settled using a number of underlying interest rate assets. One such instance is the hedging of US Treasury futures. In such a case the hedger needs to estimate which of the available bonds is likely to be the cheaper to delivery at expiry. With this assumption the hedge can be evaluated in a normal fashion. If before the expiry of the contract the interest rate environment changes such that a different bond is likely to be delivered then the hedge will need to be adjusted accordingly.

Convexity

Duration based hedging though straight-forward to apply will not lead to a perfect hedge. Further, the hedge produced will strictly only apply to parallel shifts in the interest rate curve. For non-parallel shifts of the interest rate curve duration based methods will give a good approximation for small shifts of the interest rate. For larger shifts non-linear effects of the interest rate curve will render the duration an impractical guide for effective evaluation and hedging applications.

For exactly the same reason as the gradient of a smooth functions tangent is locally approximately equal to the function itself. So the price of an interest dependent instrument for small shifts in the interest rate depends almost entirely on the duration. But for large shifts in the interest rate curve the duration will not account even approximately for the total change in the assets price.

In order to account for the non-linear dependence on interest rates (for large shifts) we consider the ‘next term of the expansion’ of the pricing function and introduce the notion known as convexity. The convexity is formally given by:

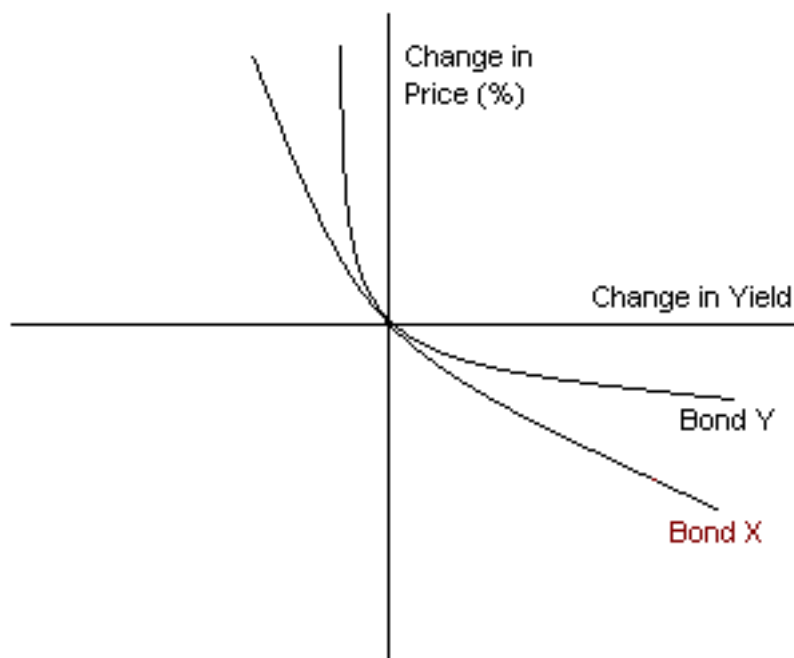
$$C = \frac{1}{B} \frac{\partial^2 B}{\partial y^2} = \frac{\sum_{i=1}^n p_i t_i^2 \exp(-y t_i)}{B}$$

where B is the assets price, p_i is the i th interest payments at time t_i and y is the yield of the investment. Then the percentage change in the interest dependent asset price of shifts of the interest rate curve taking into account the convexity of the asset is given by:

$$\frac{\Delta B}{B} = -D\Delta y + \frac{1}{2}C\Delta y^2$$

Below we illustrate the effect which differing convexities can have on the bonds price/yield ratio.

Two bonds X and Y with differing convexities



Chapter 4

Yield of Fixed-Interest Bonds on Interest Payment Dates

Within this chapter we consider various bond metrics of Fixed-Interest Bonds which exhibit closed formulae. A Fixed-Interest bond is a bond which pays a fixed income (known as the coupon or interest payment) usually at regular intervals during the life of the bond.

We cover simple yield to maturity (JGB), gross/net redemption yield, current price, holding period return (US and UK), duration and yield curves according to expectation theory. To simplify our treatment we assume where appropriate that bond metrics are evaluated on interest payment dates.

Remark In order to evaluate many of the bond metrics within this section it has been necessary to solve a rational polynomial expression. In order to solve these expressions we apply the Bisection and Newton Raphson methods (Click [here](#) for more details).

4.1 Simple Yield to Maturity (for Japanese Government Bonds (JGB))

In Japanese bond markets, the concept of simple interest is used when calculating the yield to maturity (simple yield to maturity) rather than the usual compound interest method (redemption yield). The simple yield to maturity is given by:

$$\text{Simple yield to maturity} = \frac{D}{P} + \left(\frac{100 - P}{P} \times \frac{1}{n} \right) \quad (4.1.1)$$

where,

- P = current price (including accrued interest)
- D = annual coupon payable in half-yearly installments
- n = outstanding term to redemption in years expressed in decimal format (i.e. 3 years 6 months = 3.5)

Our Implementation

Within our EJB Component we implemented the Simple Yield to maturity within:

- FixedInterestBonds.simpleYieldToMaturity(double,double, double)

Notation Suppose that the rate of interest is i per interval. Then the present value a_n , of this series of payments is given by:

$$a_n = \frac{1}{(1+i)} + \frac{1}{(1+i)^2} + \cdots + \frac{1}{(1+i)^n} = \frac{1 - (1+i)^{-n}}{i} \quad (4.1.2)$$

From now on we denote by a_n , a series of payments of one per interval payable in arrears for n intervals.

4.2 Gross Redemption Yield and Current Price

Using the same notation as above, for an interest payment date the gross redemption yield is defined as $2i$ where:

$$2i = \frac{D}{P} + \left(\frac{100 - P}{P} \times \frac{2i}{(1+i)^{2n} - 1} \right) \quad (4.2.3)$$

Remark It is usual in the US and the UK to calculate and express redemption yield as a yield per annum, convertible half-yearly.

If there is less than half a year to the next coupon payment, then the equation of value at the present time is:

$$P = \frac{1}{(1+i)^{2l}} \left(D_1 + \frac{D}{2} a_{2m} + \frac{100}{(1+i)^{2m}} \right) \quad (4.2.4)$$

where, D_1 is the next interest payment, D is the annum coupon paid in half-yearly installments, l is the period to next payment (in years), m is the period from the next interest payment to the redemption date (in years) and a_{2m} is given by (4.1.2).

Remark For a UK gilt which are ex-dividend, we can set $D_1 = 0$, because the next interest payment will not be received by a buyer.

Algorithm used to solve these equations

In order to find the solutions $2i$, respectively i of the above equations (4.2.3), (4.2.4); we have used a combination of the Bisection and Newton-Raphson methods for finding the roots of an equation of one variable. After choosing an interval in which i lies (in almost all instances i will lie in the interval $(0, 1)$), our algorithm takes a Bisection step whenever Newton-Raphson would take the solutions out of the chosen bounds, or whenever Newton-Raphson “doesn’t converge rapidly enough”.

Our Implementation

Within our implementation we have implemented methods by which the above two formulae can be solved (and hence the yield found) by application of a combination of the Bisection and Newton-Raphson methods. In particular, we have provided the following:

- `FixedInterestBonds.grossRedemptionYield(double, double, int)` - Calculates the gross redemption yield of a fixed-interest bond on an interest payment date with an integer number of years until maturity by solving the equation (4.2.3) by applying a combination of the Bisection and Newton-Raphson methods. That is, we evaluate the total (annual) yield of the fixed interest bond from now until maturity where the tax implications of the investor are not taken into account.
- `FixedInterestBonds.grossRedemptionYield(double, double, int, double, double)` - Offers the same functionality as the method listed directly above except that here we are able to specify the algorithm specific equation solving parameters such as the upper bound of the solution and the precision used.
- `FixedInterestBonds.grossRedemptionYield(double, double, double, int, double)` - Calculates the gross redemption yield if there is less than half a year to the next coupon payment of the fixed interest bond by solving the equation (4.2.4) via a combination of the Bisection and Newton-Raphson methods.
- `FixedInterestBonds.grossRedemptionYield(double, double, double, int, double, double, double)` - Offers the same functionality as the method listed directly above except that here we are able to specify the algorithm specific equation solving parameters, namely the upper bound of the solution and the precision by which the result is returned.

We also implement a formulae which relates the gross redemption yield to the price of a fixed interest bond, namely the method:

- `FixedInterestBonds.currentPrice(double, double, int)`

4.3 Net Redemption Yield (excluding CGT)

For an interest payment date, the net redemption yield is $2i$ where i is a solution of the following equation:

$$P = (1 - t) \frac{D}{2} a_{2n} + \frac{100}{(1 + i)^{2n}} \quad (4.3.5)$$

where t is the investor's rate of tax on income expressed in decimal format (i.e. 1 percent = 0.01), D is the annual coupon paid in half-yearly installments and a_n is given by (4.1.2).

Remark In short, the net redemption yield is just the gross redemption yield minus

the effect of the investors income tax on the investment income.

Algorithm used to solve these equations

In order to find the solution $2i$, respectively i of the equation (4.3.5) we have used a combination of the Bisection and Newton-Raphson methods for finding the root of an equation of one variable. After choosing an interval in which i lies (in almost all instances i will lie in the interval $(0, 1)$), our algorithm takes a Bisection step whenever Newton-Raphson would take the solutions out of the chosen bounds, or whenever Newton-Raphson “doesn’t converge rapidly enough”.

Our Implementation

Within our implementation we have implemented methods by which the above formula can be solved (and hence the yield found) by application of a combination of the Bisection and Newton-Raphson methods. In particular, we have provided the following:

- `FixedInterestBonds.netRedemptionYield(double, double, int)` - Calculates the net redemption yield of a fixed-interest bond on an interest payment date with an integer number of years until maturity by solving the equation (4.2.3) via a combination of the Bisection and Newton-Raphson methods. That is, we evaluate the total (annual) yield of the fixed interest bond from now until maturity where the income tax implications from the investor are taken into account.
- `FixedInterestBonds.netRedemptionYield(double, double, int, double, double)` - Offers the same functionality as the method listed directly above except that here we are able to specify the algorithm specific equation solving parameters, namely the upper bound of the solution and the precision by which the result is returned.

4.4 Holding Period Return (US and UK)

The holding period return is the yield over the period that the bond was held by the investor. Here we are able to evaluate the holding period return on an coupon payment date. That is, on an interest payment date the holding period half-yearly return j is given by the following equation:

$$P = \frac{D}{2} \left(\frac{1 - (1 + j)^{-2n}}{j} \right) + \frac{S}{(1 + j)^{2n}} \quad (4.4.6)$$

where P is the purchase price, D is the annual coupon payable in half-yearly installments, n is the (whole) number of years for which the bond is held and S is the market price at which the investor can sell the bond.

Remark The holding period return is used widely within the UK and US bond markets.

Algorithm used to solve these equations

In order to find the solution j , of the equation (4.4.6) we have used a combination of the Bisection and Newton-Raphson methods for finding the root of an equation of one variable. After choosing an interval in which j lies, our algorithm takes a Bisection step whenever Newton-Raphson would take the solutions out of the chosen bounds, or whenever Newton-Raphson “doesn’t converge rapidly enough”.

Our Implementation

Within our implementation we have implemented methods by which the above formula (4.4.6), can be solved (and hence the holding period return found) by application of a combination of the Bisection and Newton-Raphson methods. In particular, we have provided the following:

- `FixedInterestBonds.holdingPeriodReturn(double, double, double, int)` - Calculates the (half-yearly) holding period return of a fixed-interest bond on an interest payment date when the bond is held for a (whole) number of years by solving the equation (4.4.6) via a combination of the Bisection and Newton-Raphson methods.
- `FixedInterestBonds.holdingPeriodReturn(double, double, double, int, double, double)` - Offers the same functionality as the method listed directly above except that here we are able to specify the algorithm specific equation solving parameters, namely the upper bound of the solution and the precision by which the result is returned.

4.5 Bisection and Newton Raphson

Throughout our treatment of fixed-interest bonds we have been required to find solutions to polynomial equations. Generally speaking, this cannot be done analytically and so we have relied upon the Bisection and Newton-Raphson methods which are detailed below. First we briefly remark on bracketing a solution to an equation.

4.5.1 Bracketing One Dimensional Equations

If you are given an arbitrary function then there is no certain way of deciding a range in which a solution exists or even of determining if it has roots at all.

Notation We say that a root is *bracketed* in the interval (a, b) if $f(a)$ and $f(b)$ have opposite signs.

If the function is continuous then within a bracketed region there must exist at least one root by the intermediate value theorem. For a discontinuous function we cannot guarantee the existence of a root but from a numerical point of view the behavior is indistinguishable

from the continuous case. That is singular points where the function changes sign are identical to real roots for a computer based algorithm.

Remarks

- To find a bracketed interval we proceed by taking different points and just evaluating the function. Naturally, for a given problem the bracketed interval may be known a priori.
- Once we know an interval in which a solution is known to exist we can apply several classical procedures which should converge to a root. These proceed with various rates and sureness to a root.

4.5.2 Bisection Method

Given a bracketed interval the bisection method will always converge to a solution. Given an interval (x_1, x_2) we choose a point halfway between x_1 and x_2 , and evaluate the function. We replace either x_1 or x_2 depending upon which point given the same sign when the function is evaluated there. And so on...

Clearly after n iterations the root is known to be within an interval of size ϵ_n where:

$$\epsilon_n = \frac{|x_1 - x_2|}{2^n} \quad (4.5.7)$$

Thus, we have:

$$n = \log_2 \frac{|x_2 - x_1|}{\epsilon} \quad (4.5.8)$$

where ϵ is the desired ending tolerance.

4.5.3 Newton-Raphson Method

The Newton-Raphson method is the best known method for finding the roots of an equation. This procedure can be generalized in various directions, in particular it can be applied to finding solutions of non-linear equations. The reader should note that the assumptions required by the Newton-Raphson method differ from the Bisection method in the following ways:

- The solution does not need to be bracketed but a point sufficiently close to a solution must be chosen
- The derivative of the interpolated function needs to be evaluated at a sequence of points
- The method may not converge if the initial point is not sufficient close to a solution or the interpolated function has certain pathological features such as it oscillates very quickly close to a solution

Assuming that an initial estimate x_0 of the solution to an equation $f(x) = 0$ is known, the Newton-Raphson method will produce a sequence of values $\{x_n : n \in \mathbb{N}\}$, which will should converge. In particular, when $f(x)$ is a rational function and x_0 is chosen sufficiently well this sequence will always converge. If the algorithm fails to give a solution which in the case of polynomials means $x_n \rightarrow \pm\infty$ or the sequence is stuck in something like a loop, then we repeat the process with another initial point. By calculating the derivative $f'(x)|_{x=x_0}$ at the point x_0 , we draw the tangent to the curve at x_0 and then trace this line down to the x -axis which will give us our second point x_1 . Repeating this process will result in the desired convergent sequence. In short, we use this iterative formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n \geq 0 \quad (4.5.9)$$

Chapter 5

Interest Calculations

Since calculations related to interest payments play such a fundamental role in the theory of bonds we have decided to include an entire chapter (and corresponding software module) which covers various topics related to interest calculations. Within this chapter we detail the functionality relating to the interest bearing investments which has been implemented within the interest module.

5.1 Preliminaries and Notation

5.1.1 Preliminaries

The notions of compound and simple interest form the foundations of investment mathematics. These notions are so central because money has a ‘time value’ because it earns interest. That is, if left to compound the money earns interest on the principal sum and also interest on the interest.

The related mathematical objects to interest are those of exponents, arithmetic series and geometric series. Below we summarize the essential formulae of exponents, arithmetic series and geometric series so that a developer may more modify our interest relations if they do not fit your exact requirements.

Exponents

Given any positive numbers x, y , and real numbers n, m , we have the following law of exponents:

- $x^0 = 1, (-x)^0 = 1$
- $(-x)^n = (-1)^n x^n$
- $x^n x^m = x^{n+m}$
- $\frac{x^m}{x^n} = x^{m-n}$

- $(x^m)^n = x^{mn}$
- $(xy)^n = x^n y^n$
- $\left(\frac{x}{y}\right)^n = \frac{x^n}{y^n}$

Arithmetic Series

An arithmetic series A_n , is given by:

$$A_n = a + (a + b) + (a + 2b) + \dots + (a + (n - 1)b) \quad (5.1.1)$$

where $a \in \mathbb{R}$, $R \in \mathbb{R}$ the common difference and $n \in \mathbb{N}$ is the number of term in the series. By induction it follows that the sum of this series is given by:

$$A_n = an + \frac{n(n - 1)b}{2} \quad (5.1.2)$$

Geometric Series

A finite geometric series S_n , is given by:

$$S_n = b + bR + bR^2 + bR^3 + \dots + bR^{n-1} \quad (5.1.3)$$

where $b \in \mathbb{R}$, $R \in \mathbb{R}$ the common ratio and $n \in \mathbb{N}$ is the number a terms in the series. By induction it follows that the sum of this series is:

$$S_n = \frac{b(1 - R^n)}{(1 - R)} \quad (5.1.4)$$

An infinite geometric series is an indefinite sum of terms is denoted by:

$$S = b + bR^2 + bR^3 + \dots \quad (5.1.5)$$

where ‘...’ denotes that the sum extends in a similar fashion indefinitely. If the modulus (i.e. absolute value) of R , is greater than one then the series does not have a finite sum and is said to diverge. If the modulus a R is strictly less than one then the infinite sum will converge to some finite value. That is, the sum will get closer and closer to some finite value as we sum more and more terms. The sum of such an infinite series is given by:

$$S = \frac{b}{1 - R} \quad (5.1.6)$$

5.1.2 Notation

Throughout this documentation we will use the following notation:

- p = number of periods of time

- $d(0)$ = initial deposit or investment
- $d(t)$ = accumulated values after t periods of time
- $d_r(t)$ = the real worth of the investment after t periods of time
- i = rate of interest expressed in decimal form, i.e. $1\% = 0.01$
- $i^{(p)}$ = p -yearly rate of interest
- $j(p)$ = effective interest
- r = rate of depreciation expressed in a decimal form, i.e. $1\% = 0.01$
- s_n = accumulated value of an annuity certain
- a_n = present value of an annuity certain
- f = rate of inflation (annual) expressed in decimal form, i.e. $1\% = 0.01$

5.2 Simple Interest

5.2.1 Basic formula

The simple interest is calculated using the following formula:

$$d(1) = d(0)(1 + i) \quad (5.2.7)$$

That is, we calculate $d(1)$ for i and $d(0)$ given.

Example If \$200 is deposited into a bank account with a yearly interest rate of 5%, after 1 year the accumulated sum (in \$) is:

$$d(1) = 200(1 + 0.05) = 210 \text{ dollars}$$

5.2.2 Real worth

The real worth $d_r(1)$, of a fixed interest bearing investment over a given period where the initial deposit was $d(0)$ is given by:

$$d_r(1) = \frac{d(0)}{1 + f} (1 + i) \quad (5.2.8)$$

where f is the ('constant') rate of inflation over the period and i is the (fixed) rate of interest over the period.

5.2.3 Real return

The real return j , of an interest bearing investment over a given period is given by:

$$j = \frac{i - f}{1 + f} \quad (5.2.9)$$

where i is the fixed interest over the period and f is the ('fixed') rate of inflation.

We also provide methods which calculate/estimate the average inflation and interest rate over a number of periods.

5.3 Compound Interest

5.3.1 Accumulated values

The accumulated value $d(p)$ of an interest bearing investment $d(0)$, after p periods of time where the fixed rate of interest is i , is given by:

$$d(p) = d(0) (1 + i)^p \quad (5.3.10)$$

We also provide a method which calculates the amount which an investor must deposit in order to receive a given sum after a certain number of years.

Example If an investor wishes to receive \$10,000 from an fixed interest bearing investment which pays 5% after 5 years then they must deposit today:

$$\frac{10000}{(1 + 0.05)^5} = \$7,835.26$$

5.3.2 Real worth

The real worth of an fixed interest bearing investment after p periods of time, where the annual rate of inflation is f and the initial deposit is $d(0)$. The fixed interest rate is i per interval of time where each interval consists of n days. We provide formulae for the real worth according to the day count conventions in Europe, UK and Japan for the quotation of the annual rate of inflation:

- Europe

$$d_r(p) = \frac{d(0)}{(1 + f)^{\frac{np}{360}}} (1 + i)^p \quad (5.3.11)$$

- UK and Japan

$$d_r(p) = \frac{d(0)}{(1 + f)^{\frac{np}{365}}} (1 + i)^p \quad (5.3.12)$$

We also provide the general form when the annual inflation is quoted with respect to m days. Then the real worth is given by:

$$d_r(p) = \frac{d(0)}{(1+f)^{\frac{np}{m}}} (1+i)^p \quad (5.3.13)$$

We also consider in the general case the problem of determining the initial deposit necessary in order to get a pre-determined real worth from a fixed interest bearing investment. If the annual inflation is quoted with respect to m days and the final real worth of the investment after p periods is required to be M then the initial deposit must be:

$$\frac{M(1+f)^{\frac{np}{m}}}{(1+i)^p}$$

5.3.3 Real return

The real return j of an interest bearing investment is over p time periods is calculated where the annual rate of inflation f is assumed to be constant and the interest rate is i per period. We assume that each time period has n days, and the annual inflation is quoted in accordance with European, UK and Japanese conventions:

- Euro

$$j = \frac{(1+i)^p}{(1+f)^{\frac{np}{360}}} - 1 \quad (5.3.14)$$

- UK and Japan

$$j = \frac{(1+i)^p}{(1+f)^{\frac{np}{365}}} - 1 \quad (5.3.15)$$

We also provide the general form where the annual inflation is quoted with respect to m days. Then the real return is given by:

$$j = \frac{(1+i)^p}{(1+f)^{\frac{np}{m}}} - 1 \quad (5.3.16)$$

We also provide the general form when the annual inflation is quoted with respect to m days. then the real return is given by:

$$j = \frac{1+i}{(1+f)^{\frac{n}{m}}} - 1 \quad (5.3.17)$$

5.3.4 Depreciation

Physical assets which are owned by a corporation (e.g. cars) often decrease in value with time. This process is referred to as depreciation. Estimates of the rate of depreciation are useful since for accounting purposes this unrealized loss can be set against a companies profits.

The depreciation rate is expressed as the percentage decrease in the assets market value over a period of time. We offer methods which calculate the percentage decrease in the book value of a collection of assets over one or many time periods.

If P_0 is an assets value (often referred to as book value) at time 0 and P_t is the assets value (or expected value) after t periods of time then the **depreciation rate per time period** is given by:

$$P_t = P_0 (1 - r)^t \quad (5.3.18)$$

Example: The cost of an asset is known to be 20,000 and its salvage value is estimated at 8,000 after a useful life of 5 years.

$$\text{Depreciation of asset} = 1 - \left(\frac{8,000}{20,000} \right)^{\frac{1}{5}}$$

We also provide a method which calculates the book value of an asset after a number of time periods in which the depreciation rate may differ. If the book value P_0 , of an asset declines with a **varying rate**, r_i being the rate of depreciation of the i th period we will obtain the book value P_t at the end of t years:

$$P_t = P_0 (1 - r_1) (1 - r_2) \cdots (1 - r_t) = P_0 \prod_{i=1}^t (1 - r_i) \quad (5.3.19)$$

5.3.5 Force of interest

If the nominal rate is continuously compounded from p to infinity, the resultant annual convertible rate of interest is referred to as the force of interest and is denoted by the symbol δ . We calculate δ using the following expression:

$$\exp(1)^\delta = 1 + i$$

where $\exp(1)$ denotes the exponential function evaluated at 1 (i.e. 2.718...).

Therefore,

$$\delta = \ln(1 + i) \quad (5.3.20)$$

If the force of interest δ is quoted then the following relation can be useful:

$$d(t) = d(0) \exp(1)^{t\delta} \quad (5.3.21)$$

5.4 Effective and Nominal Interests

5.4.1 Effective rate of interest

We convert the effective p -yearly rate into the q -yearly rate. If $j(p)$ is the p -yearly rate, then the q -yearly rate of interest is $j(q)$, where:

$$(j(p) + 1)^p = (j(q) + 1)^q$$

That is,

$$j(q) = (j(p) + 1)^{\frac{p}{q}} - 1 \quad (5.4.22)$$

but this is only the interest over the q^{th} period. Therefore, the interest compounded over the q^{th} periods to give the annual rate is:

$$j(q) = q(j(p) + 1)^{\frac{p}{q}} - q \quad (5.4.23)$$

5.4.2 Real return

We calculate the real return j_q of a q -yearly rate, knowing the p -yearly rate and the annual rate of inflation:

$$j_q = q \frac{\left(1 + \frac{i^{(p)}}{p}\right)^{\frac{p}{q}}}{(1 + f)^{\frac{1}{q}}} - q \quad (5.4.24)$$

5.4.3 Nominal rate of interest

If the effective rate of interest is expressed in terms of $1/p$ of a year, it is often converted to an annual rate by simply multiplying by p . The rate of interest i per annum is p -yearly convertible. The rate of interest quoted in this way is known as the nominal rate of interest. If the effective annual rate is i , the nominal rate of interest convertible p -yearly is $i^{(p)}$, where:

$$\left(1 + \frac{i^{(p)}}{p}\right)^p = 1 + i$$

5.5 Net Present value

To calculate the initial amount of money $d(0)$ which has to be invested in order to accumulate the final sum $d(t)$, where t is the number of periods and i is the interest over each of these periods (assumed to be constant), we use:

$$d(0) = \frac{d(t)}{(1 + i)^t} \quad (5.5.25)$$

Remark The constant t need not to be an integer.

The initial amount of money $d(0)$, which needs to be invested in order to accumulate a final worth $d_r(t)$, after t periods of time, if the annual rate of inflation is f and the period of time has p days:

$$d(0) = \frac{(1 + f)^{\frac{pt}{360}} d_r(t)}{(1 + i)^t} \quad (5.5.26)$$

where i is the interest rate paid over each period.

Remark Present values provides a method of comparing monies available at different

points in time. The calculation of the present value of future sums of money is referred to as *discounting*.

Example: You need to decide between two business opportunities. The first opportunity will pay \$700 in 4 year's time and the second opportunity will pay \$850 in 6, year's time. You have been advised to discount these future sums by using the interest rate of 8%.

By substitution we are able to calculate a present value for each of these future sums, which gives:

- First opportunity: $d_0 = 514.52$
- Second opportunity: $d_0 = 535.64$

Therefore, the second business opportunity is preferable to the first opportunity.

5.6 Annuities

An annuity is a series of periodic payments (or cash flows) of equal size. Annuities occur within a number of different investment scenarios including:

- Bond Coupon's
- Social Security Payments
- Retirement Plan payments

Within this section we consider how the accumulated value and the present value of an annuity is evaluated.

5.6.1 Accumulated Values of an Annuity

The accumulated value

The accumulated value of a series s_n of payments of one per interval, payable in arrears for n intervals (where i is the interest and n is the number of intervals) is:

$$s_n = \frac{(1+i)^n - 1}{i} \quad (5.6.27)$$

Series of payments

Series of payments of one per interval payable in advance for n intervals. Calculate the accumulated value S_n of this series of payments:

$$S_n = (1+i)s_n \quad (5.6.28)$$

5.6.2 Present Value of Annuities

The following types of annuity's are particularly important in investments and it is therefore useful to know the general formulas for their present values.

Series of payments in arrears

Series cash of payments of one per intervals payable in arrears for n intervals

Denoted by the symbol a_n . The fixed rate of interest i is given. The present value a_n , is given by:

$$a_n = i^{-1} - (1 + i)^{-n} \quad (5.6.29)$$

where i is the interest rate paid on cash.

Remark In the case of large investment banks the interest rate paid on cash is known as the risk free interest rate which we be approximately equal to the yield paid on near dated locally denominated government bonds.

Series of payments in advance

Series of cash payments of one per interval payable in advance for n intervals

The present value of this series is denoted by the symbol A_n , and is given by:

$$A_n = (1 + i)a_n \quad (5.6.30)$$

where i in the interest rate paid on cash.

Infinite series of payments in arrears

Infinite series of cash payments made at the end of each interval The initial payment is d_1 and each subsequent payment is $(1 + g)$ times the previous payment.

The present value is given by:

$$\text{Present value} = \frac{d_1}{i - g} \quad (5.6.31)$$

where i in the interest rate paid on cash.

Remarks

- The present value converges if and only if i is greater than g . Hence, our method throws an exception if g is less than or equal to i since clearly this situation cannot occur.
- When $g = 0$ (i.e. there is zero growth) the present value of an infinite series of payments known as *perpetuity*.

Increasing annuity

An increasing annuity in which the first payment is one after one interval, the second payment is two after two intervals and so on... with a final payment of n after n intervals. The present value of this series is denoted by $(Ia)_n$, where:

$$(Ia)_n = \frac{A_n - nv^n}{i} \quad (5.6.32)$$

where $v = \frac{1}{1+i}$.

5.6.3 Series of multiple payments per interval in arrears

Series of payments of one per interval payable p times per interval in arrears for n intervals. If the constants n, p, i are given then the present value is denoted by $a_n^{(p)}$, is given by:

$$a_n^{(p)} = \frac{i}{i^{(p)}} a_n \quad (5.6.33)$$

Series in multiple payments per interval in advance

Series of payments of one per intervals payable p times per intervals in advance for one interval. Given the constants i, p and n , the present value is denoted by $A_n^{(p)}$ and is given by:

$$A_n^{(p)} = \left(\frac{i}{i^{(p)}} + \frac{i}{p} \right) a_n \quad (5.6.34)$$

5.7 Yield (internal rate)

If an investment X provides X_p money at the end of n years, then the equation of value is:

$$X(1+i)^n = X_p \quad (5.7.35)$$

The rate of interest for which this equation of value is satisfied is the yield.

5.7.1 Finding the yield

Let $a(k)$ be a function defined on the natural numbers $\mathbb{N}$ with values in $\{0, 1\}$, $a(n) = 0$ if the payment has not been performed in the n -th period of time and $a(n) = 1$, if the payment has been performed in the n -th period of time.

Assuming that the function $a(k)$ for $\{k \in \mathbb{N} : 1 \leq k \leq n\}$, and i the fixed interest rate are known we can calculate the yield j , using the following formula:

$$a(1)(1+i) + a(2)(1+i)^2 + \cdots + a(n)(1+i)^n = (1+j)^n \quad (5.7.36)$$

Alternatively, if the initial investment $d(0)$ and the amount repaid after n periods of time $d(n)$ is known. We can calculate the yield j , by:

$$d(0)(1+j)^n = d(n) \quad (5.7.37)$$

5.7.2 The relationship between real and nominal yield

If f is the annual rate of inflation and i is the effective annual nominal yield. Then the real yield j is given by:

$$j = \frac{i - 1}{1 + f} \quad (5.7.38)$$

5.8 Repo agreements

A repo agreement is an agreement between two parties where one party agrees to sell and then buy back an asset at a later date from the counter party.

Remark The advantage of such an agreement is that it allows a party to temporarily borrow cash from a counter party who if the contract to structured correctly assumes very little risk.

We denote:

- f = the inflation during the period of the repo agreement. For example, if the annual inflation runs at 6% during a six month repo agreement then the inflation during the period is 3%
- x = initial purchase/sale price when the repo contract is entered into
- X_f = final payment made at the end of the repo agreement
- X = final worth of the investment

Then the real return j , from a repo agreement is given by:

$$x(1 + j) = X = \frac{X_f}{1 + f}$$

Hence, the real return from the repo agreement is given by:

$$j = \frac{X_f}{(1 + f)x} - 1 \quad (5.8.39)$$

Chapter 6

Programmer's Guide

This chapter provides technical documentation for developers implementing J2EE Applications using the WebCab Bonds (J2EE Edition). Its purpose is to enable the efficient exploitation of all methods and related features and to enhance the use of future WebCab J2EE Applications.

6.1 Introduction to Fixed-Interest Bonds Component

The WebCab Fixed-Interest Bonds J2EETM Application performs the following calculations:

- Redemption Yield
 - **Gross Redemption Yield** - Knowing the current price, the coupon and the number of years. Note if the method **grossRedemptionYield()** did not find a solution for the problem, you have to add two parameters to the method: one is the upper limit for the solution and the other is the accuracy of the solution.
 - **Gross Redemption Yield** - If there is less than half a year to the next coupon payment - knowing the current price, the coupon, the next interest payment, the period for next interest payment to redemption (in years) and the period to the next payment. Note if the method **grossRedemptionYield()** did not find a solution for the problem, you have to add two parameters to the method: one is the upper limit for the solution and the other is the accuracy of the solution.
 - **Net Redemption Yield** - Knowing the current price, the coupon, the investor's rate of tax on income and the number of the intervals. Note if the method **grossRedemptionYield()** did not find a solution for the problem, you have to add two parameters to the method: one is the upper limit for the solution and the other, the accuracy of the solution.
- Interest Yield
 - Gross Interest Yield
 - * Before expenses - Knowing the clean price and the coupon.

- * After expenses, in case of a purchase - Knowing the clean price and the coupon.
 - * After expenses in case of a sale - Knowing the clean price and the coupon.
- Net Interest Yield
 - * before expenses - Knowing the clean price, the coupon and the investor's rate of tax on income.
 - * after expenses, in case of a purchase - Knowing the clean price, the coupon and the investor's rate of tax on income.
 - * after expenses, in case of a sale - Knowing the clean price, the coupon and the investor's rate of tax on income.
- Holding Period Return - Knowing the purchase price, the coupon, the price the investor's sold bond and the number of years of which the bond is held. Note if the method **holdingPeriodReturn()** did not find a solution for the problem, you may have to add two parameters to the method: one is the upper limit for the solution and the other is the accuracy of the solution.
- Japanese Government Bonds - Simple Yield to Maturity - Knowing the current price including accrued interest, the coupon payable in half yearly installments and the outstanding term of redemption (in years).
- Duration - knowing the current price including accrued interest, the coupon and the gross redemption yield i or the value $v = \frac{1}{1+i}$.
- Current Price
 - Knowing the coupon and the gross redemption yield.
 - If there is less than a half a year to the next coupon payment, calculates the current price knowing the coupon, the gross redemption yield, the next interest payment, the period for the next interest payment to redemption (in years) and the period to next payment.
- Series of Payments - Knowing the rate of interest per interval and the number of intervals.
- Rate of Payments - Knowing the series of payments of one per interval payable in arrears for a number of intervals.
- Gross Redemption According to Expectation Theory - knowing the series of assumed short term interest rates expected by the market.

6.2 Client-Side Implementation Procedure

To make a client program that uses the methods provided by this component, you will need to create a new instance of the component's remote interface.

The following code provides a standard way to opening a connection to an Enterprise JavaBeansTM Application Server and making preparations for subsequent invocations of the bean's methods and fields.

```
import com.webcab.ejb.finance.fib.*;

try {
    Properties props = System.getProperties ();
    Context ctx = new InitialContext (props);

    FixedInterestBondsHome home = (FixedInterestBondsHome)
        ctx.lookup("FixedInterestBonds");
    FixedInterestBonds bean = home.create ();
}
catch (Exception e) {
    e.printStackTrace ();
}
```

6.3 Disposing Resources

When all necessary operations have been completed, you may continue your Java application by freeing both client and server-side resources, with the following lines of code.

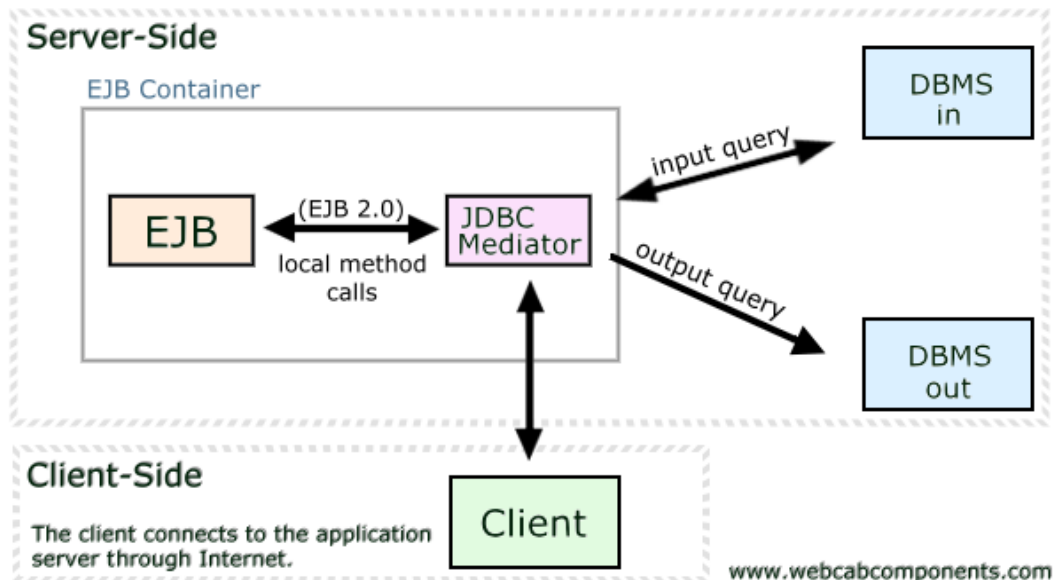
```
try {
    bean.remove();
}
catch (Exception e) {
    e.printStackTrace ();
}
```

6.4 Using JDBC Mediator with our EJBs

6.4.1 Overview

The JDBC mediator is an EJB component which mediates between an EJB component, it clients and DBMS. For each EJB component there exists a corresponding mediator with its own exception class. The mediator receives an input and an output SQL query for the DBMS and a method name to analysis. The mediator sends the SQL input query to the (input) DBMS and then applies the method from the EJB component to each row of the

input query results. Each result row is written into the (output) DBMS according to the output SQL query.



A diagram that illustrates how JDBC Mediator intermediates the communication between the client, the Application Server and the Database Servers.

6.4.2 Connecting to your Database Server

If you wish to connect by JDBC to your Database server in order to run a select query an update statement or a stored procedure you will need to describe the connection with certain properties. You will be given the opportunity to use JDBC Mediator with Application Server DataSources and Connection Pools or directly specifying these properties and passing them to the `create` methods.

- **Driver**

The driver is a Java class provided by your Database vendor that implements the `java.sql.Driver` interface.

- **URL**

This property defines the address of your DBMS, the port number and additional information such as service and database name.

- **Username and Password**

For authentication reasons, most connections will not work without providing a user name and a password.

- **Additional Properties**

This field allows you to customize even more your JDBC connection as described in the JavaDocs

We describe these properties individually for the most well known database systems. Please feel to jump to the section which refers to the particular DBMS you plan to use with our JDBC components.

Oracle 9i

We recommend the JDBC thin driver `oracle.jdbc.driver.OracleDriver`. You may download this driver from http://otn.oracle.com/software/tech/java/sqlj_jdbc/content.html. The Oracle URL has the following format:

```
jdbc:oracle:thin:@servername:port:service
```

where *servername* is the Internet name of your Oracle DBMS, *port* is usually 1521 and *service* is the name of the service you are going to connect to.

IBM DB2

The name of the DB2 driver is `COM.ibm.db2.jdbc.net.DB2Driver` and can be downloaded off the IBM site at <http://www-106.ibm.com/developerworks/db2/zones/java/>. The format of the DB2 URL is:

```
jdbc:db2://[servername]:[port]/[database]
```

where *servername* is the Internet name of your DB2 DBMS, *port* may be ignored and *database* is the name of the database you plan to connect to.

Microsoft SQL Server

The name of the SQL Server driver is `com.microsoft.jdbc.sqlserver.SQLServerDriver` and can be downloaded off the Microsoft site at <http://msdn.microsoft.com/...>. The format of the Microsoft SQL Server URL is:

```
jdbc:microsoft:sqlserver://servername:1433
```

where *servername* is the Internet name of your SQL Server.

Sybase

The name of the Sybase driver is `com.sybase.jdbc.SybDriver` and can be downloaded off the Sybase site at <http://www.sybase.com/home>. The format of the Sybase JDBC URL is:

```
jdbc:sybase:Tds:servername:port/
```

where *servername* is the Internet name of your Sybase DBMS and *port* is the port number where the Sybase server is running.

6.4.3 JDBC Components

Most methods implemented by our enterprise Beans accept numeric parameters and return numbers or arrays of numbers. By using these JDBC methods directly inside a database server without any change in functionality will definitely prove necessary. In this case the database server becomes the DataSource for the input and/or output parameters for every one of these methods.

What we are going to discuss applies only to the following modules included in this application. Modules not listed here either do not implement JDBC or use a particular JDBC implementation.

- The “**Basic Bonds**” J2EE Module
- The “**Interest**” J2EE Module
- The “**Interest Derivatives**” J2EE Module

For every bean implementing number-based methods (methods with numeric parameters and numeric return values) there is a bean in a sub-package called “jdbc” bearing a similar name. For example, if there’s a bean called “FinancialBean” within a package called “com.webcab.ejb.finance”, the corresponding JDBC-implementing component would be “com.webcab.ejb.finance.jdbc.FinancialBeanJDBC”.

The JDBC components are designed to easily transfer to an Application Server JDBC specific tasks that make use of the capabilities of every bean within our enterprise application. The enterprise database performs demanding transactions which take place on the same machine that hosts your EJB components. This allows the client to take advantage of the processing power of your Application Server and minimize bandwidth transfer.

The JDBC beans are created by specifying all necessary JDBC connection parameters, such as drivers, url, username and password plus the enterprise Bean’s creation parameters. Every JDBC component contains several methods that invoke methods from their corresponding enterprise Bean and apply them directly to your database tables and/or write the answer back into the same or another DBMS. The JDBC methods accept the name of the enterprise method as the first parameter and an SQL Query or an array of Java objects for the method’s input and output.

The following source code listing exemplifies how to use an input/output SQL query JDBC method for a generic bean called “com.webcab.ejb.finance.FinancialBean” and a generic method “double calculateAmount (double, double)”.

```
import com.webcab.ejb.finance.FinancialBean;
import com.webcab.ejb.finance.jdbc.FinancialBeanJDBC;
...
try {
    ...
    /*
     * The first creation parameter (riskFreeInterestRate) is FinancialBean
     * specific. The next creation parameters define the input and the output
     * DBMS connections by driver, url, username, password and additional
     * properties.
     */
    FinancialBeanJDBC jdbcBean = home.create (riskFreeInterestRate,
        "oracle.jdbc.driver.OracleDriver", // Input Connection
        "jdbc:oracle:thin:@inputserver.com:1521:ORACLE",
        "SCOTT", "tiger", null,
        "oracle.jdbc.driver.OracleDriver", // Output Connection
        "jdbc:oracle:thin:@updateserver.com:1521:ORACLE",
        "MIKE", "wolf", null);

    jdbcBean.call ("calculateAmount",
        "SELECT C_ID, SHARES, VALUE FROM TRADES",
        "UPDATE CUSTOMER SET MONEY=? WHERE C_ID=?",
        new int[] [] {{2, 1}});
    catch (Exception e) {
        e.printStackTrace ();
    }
}
```

The first parameter of the `call` method represents the `FinancialBean` method name. The second parameter is the input query, a select query which returns three columns `C_ID`, `SHARES` and `VALUE`. The third parameter is a prepared statement that is meant to write the value returned by `calculateAmount` in the `MONEY` field and the `C_ID` primary key in the where clause. This assignment is described by the last parameter, an array of integer pairs that specifies the index of the IN parameter in the output query and the column number of the input query. Both indices start counting from 1. In our case, the second IN parameter is the second question mark in the update statement and the first input column is `C_ID`.

Every output update is made according to a primary key `C_ID`. Even though this is returned by the select statement it is not used for computation by the two-parameter “`calculateAmount`” method.

As an alternative you may wish to define two `DataSources` that point to the input and output connections described above. You may then map the corresponding resource references

defined inside the `ejb-jar.xml` deployment descriptor to the JNDI names corresponding to these `DataSources`. In this case scenario, the previous code would look like this:

```
import com.webcab.ejb.finance.FinancialBean;
import com.webcab.ejb.finance.jdbc.FinancialBeanJDBC;
...
try {
    ...
    /*
     * The only creation parameter is riskFreeInterestRate. Ignore the
     * JDBC specific parameters. The bean will be using the
     * resource references defined inside the ejb-jar.xml XML
     * to locate the Input and Output JDBC connections.
     */
    FinancialBeanJDBC jdbcBean = home.create (riskFreeInterestRate);

    jdbcBean.call ("calculateAmount",
        "SELECT C_ID, SHARES, VALUE FROM TRADES",
        "UPDATE CUSTOMER SET MONEY=? WHERE C_ID=?",
        new int[] [] {{2, 1}});
    catch (Exception e) {
        e.printStackTrace ();
    }
}
```

Note The primary key reference can be made across two different database servers a very useful feature provided by this type of JDBC methods.

6.5 Support for Developers

6.5.1 Compilation and Custom Modification of Source Code

This J2EE Application and related Java™ source files have been compiled using Sun's J2SDK1.4.2 and J2EE 1.3, 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.

6.5.2 Online Support

If you have any questions or queries concerning the use of this component then 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>

Chapter 7

Examples

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

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

7.1 Question and Answer (QA) Client Examples

7.1.1 Overview

Within this folder we offer Java client examples for the J2EE 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.

7.1.2 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.

7.1.3 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' --+...
                                                  |
                                                  + ...

```

7.1.4 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)

```

7.1.5 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.

7.1.6 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.
- 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..

7.2 Custom Examples

The following sections describe all custom-made examples.

7.3 Fundamental Theory of Bonds

We provide examples for each of the methods provided by this module.

7.3.1 Zero Rates

Within this subsection we describe the examples provided within the:

```
CalculatingZeroRatesClient.java
```

client which is contained within the Client\CalculatingZeroRates folder of this package. Within this client we provide one example for each of the methods within the `CalculatingZeroRates` class.

Zero Rate implied from Zero Bond rates

Problem: What is the zero rate of a zero bond with 2.13 years to maturity which has a market price of \$96.24 and a face value of \$100?

Solution: The gross interest yield is 0.0173..., which can be evaluated using:

```
zeroRateFromZeroBond(2.13, 100, 96.24)
```

method of the `CalculatingZeroRates` EJB Component.

Boot strap method

Problem: The market price of a coupon paying bond with 1.9 years to maturity is \$94. The coupon paying bond pays an annual coupon of \$1 which is due in 0.9 years. Further, the zero rate of 1 year and 0 years maturity is known to be 4% and 4.2% respectively. What is the zero rate of duration 1.9 years?

Solution: The zero rate of maturity 1.9 years is 0.0327..., which can be evaluated using the:

```
calculatingZeroRatesObject.zeroRateBootstrap(1.9, {1, 1}, { 0.9, 1.9 }, {4, 4.2}, 94, 100)
```

method of the `CalculatingZeroRates` EJB Component.

7.3.2 Forward Rates

Within this subsection we describe the examples provided within the:

```
ForwardRatesClient.java
```

client which is contained within the Client\ForwardRates folder of this package. Within

this client we provide one example for each of the methods within the `ForwardRates` EJB Component.

Evaluate the Forward rate

Problem: What is the forward rate corresponding to the period between 3 and 4 years. Where the 3 year zero rate is 10.8% and the 4 year zero rate is 11% ?

Solution: The forward rate is 0.115999..., and is given by the method:

```
forwardRate(0.108, 0.11, 3, 4)
```

method of the `ForwardRates` EJB Component. Note, that by 'reversing' the rates, that is evaluating:

```
forwardRate(0.11, 0.108, 3, 4)
```

the forward rate is now given by 0.10199..., and hence the order in which the interest rates are given effects the final result.

Forward Rate Agreement

Problem: What is the value of a forward rate agreement (FRA) of the following contract? The 1-year zero-coupon rate is 10% and the two-year zero coupon rate is 10.5%, further the 2-year zero coupon continuously compounded rate is 10.5%. Suppose we had entered into a FRA where we will receive a rate of 12% with annual compounding on a principle sum of \$1 million between the end of year one and the end of year two.

Solution: This contract can be evaluated by first evaluating the forward rate using the method:

```
forwardRate(0.10, 0.105, 1, 2)
```

which give a value of 11% with continuous compounding or 11.6278% with annual compounding. Now the forward rate agreement can be evaluated using the method:

```
forwardRateAgreement( 1000000, 0.12, 1, 2, 0.116278, 0.105)
```

which give a value of 3016.99.

7.3.3 Treasury Price

Within this subsection we describe the examples provided within the:

```
TreasuryPriceClient.java
```

client which is contained within the Client\TreasuryPrice folder of this package. Within this client we provide one example for each of the methods within the **TreasuryPrice** EJB Component.

Price of Bond

Problem: What is the theoretical price on a bond with principle sum of \$100, 2 years until maturity which pays in coupon a one year of \$5, and two years of \$7; where the 1 and 2 year zero rates are 5.2% and 5.7% respectively?

Solution: The price of the bond is evaluated by using the method:

```
priceOfBond(principleSum, timeToMaturity, zeroRateAtMaturity, payments, time-  
OfPayments, zeroRatesOnPaymentDates)
```

within the **TreasuryPrice** EJB Component, which gives a price of 100.218....

T-Bond Price

Problem: What is the price of a US T-bond which has a principle sum of \$100 and has 2.43 years until expiry? The bond pays coupons of \$4, and is due to make payments in 0.4, 0.9, 1.4, 1.9 and 2.4 years after which the bond is due to expire. Further note that the risk free interest rate is known to be 5%

Solution: The price of the T-Bond is 113.31, and is given by:

```
tbondPrice(100, 0.05, 2.43, couponPayments1, timeOfPayments1)
```

where:

- couponPayments1[] = {4, 4, 4, 4, 4}
- timeOfPayments1[] = {0.4, 0.9, 1.4, 1.9, 2.4}

Price of Treasury Zero bond

Problem: What is the price of a zero coupon UK gilt with a principle value of 100 pounds and a time to maturity of 3.4 years where the risk free interest rate for pounds is 6% ?

Solution: The price of zero bond is 81.54..., which can be evaluated using the method:

```
zeroTBondPrice(100, 0.06, 3.4)
```

from the `TreasuryPrice` EJB Component.

Yield to Maturity

Problem: What is the yield to maturity of a corporate bond with a principle value of \$100 and a market price of \$120 and 4.2 years to maturity? The corporate bond pays annual coupons of \$10 in 0.7, 1.7, 2.7 and 3.7 years from now until expiry.

Solution: The yield to maturity is 0.114... (or 11.4%), which we evaluate using the method:

```
yieldToMaturity(100, 120, 4.2, couponPayments3, timeOfPayments3)
```

from the `treasuryPrice` EJB Component, where:

- `couponPayments3` = {10, 10, 10, 10}
- `timeOfPayments3` = {0.7, 1.7, 2.7, 3.7}

Yield to Maturity of a zero bond

Problem: What is the yield to maturity of a (corporate or treasury) bond which has a market price of \$90, \$100, \$110, a principle sum of \$100, \$110, \$120 (respectively) and a time to maturity of either 1 or 2 years? Note, that all of these bonds pay a coupon of 5 after the first year and where appropriate also after the second year.

Solution: Using the method:

```
zeroYieldToMaturity(principleSum, marketPrice, timeToMaturity)
```

from the `TreasuryPrice` EJB Component we obtain the following results:

- Market Price \$90, Principle Sum \$100, Time to maturity 1 year, Yield to maturity 0.1053605156578262
- Market Price \$100, Principle Sum \$110, Time to maturity 1 year, Yield to maturity 0.09531017980432488
- Market Price \$110, Principle Sum \$120, Time to maturity 1 year, Yield to maturity 0.08701137698962982

- Market Price \$90, Principle Sum \$100, Time to maturity 2 year, Yield to maturity 0.052680257828913175
- Market Price \$100, Principle Sum \$110, Time to maturity 2 year, Yield to maturity 0.04765508990216244
- Market Price \$110, Principle Sum \$120, Time to maturity 2 year, Yield to maturity 0.04350568849481491

Remark In order to express the above results in the form $\alpha\%$, multiply each of the results by 100.

Yield to Maturity from Price

Problem: What is the yield from three different bonds with 4.25 years to maturity a principle sum of \$100, which all pay semiannual coupon payments of \$5 (with the first payment due immediately) and have a market price of \$71, \$86, \$92?

Solution: Using the method:

```
yieldToMaturityFromPrice(price, payments, timeOfPayments)
```

within the `TreasuryPrice` EJB Component we obtain the following results:

- The yield of the bond with a market price of \$71 is 0.212..., or 21.2%
- The yield of the bond with a market price of \$86 is 0.152..., or 15.2%
- The yield of the bond with a market price of \$92 is 0.132..., or 13.2%

Remark Note that the payment of the principle sum at the maturity of the bond is the final payment, not the last coupon payment.

Par Yield

Problem: What is the par yield of a US T-Bond where:

- principle sum = 100
- maturity = 10 years
- 10 Year Zero Rate = 5.7%
- The bond pays annual coupons
- time to coupon payments in years = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- zero curve on payment dates = 5.62, 5.62, 5.62, 5.63, 5.63, 5.64, 5.65, 5.67, 5.69, 5.7

Solution: The Par yield is 11.03%, and is evaluated using the method:

```
parYield (principleSum, maturity, annualOrSemiAnnual, maturityZeroRate, time-  
ToCouponPayments, zeroCurveOnPaymentDates)
```

from the `TreasuryPrice` EJB Component.

7.3.4 Duration

Within this subsection we describe the examples provided within the:

```
DurationConvexityClient.java
```

client which is contained within the `Client\DurationConvexity` folder of this package. Within this client we provide one example for each of the methods within the `DurationConvexity` EJB Component.

Price of Bond

Problem: What is the theoretical price of a bond which has a continuously compounded yield of 6%, has a principle sum of \$100 and pays semiannual coupons of 1 and has two years until maturity?

Solution: The price of the bond is 92.51 . . . , which we evaluated using the method:

```
priceOfBond(yield, payments, timeOfPayments, principleSum, timeToMaturity)
```

from the `DurationConvexity` EJB Component.

Duration of Bond

Problem: What is the duration of a bond with continuously compounded yield of 6%, has a principle sum of \$100 and pays a semi-annual coupon of 1 and has two years until maturity?

Solution: The duration of the bond is 0.03024 . . . , which we evaluated using the method:

```
duration(yield, payments, timeOfPayments, principleSum, timeToMaturity)
```

from the `DurationConvexity` EJB Component.

Price change under Parallel Yield Curve Shift

Problem: What is the percentage change in the price of a bond which has a duration of 0.73, when the yield curve experiences a parallel shift of 5 basis points (i.e. 0.0005%)?

Solution: The percentage price change in the price of the bond is -0.0365% , which we evaluated using the method:

```
percentagePriceChange (double deltaYield, double duration)
```

from the DurationConvexity EJB Component.

Weight within Bond Portfolio

Problem: What is the weights of each bond within a bond portfolio with 10 assets with values 1, 2, 3, ..., 9, 10.

Solution: The weights can be evaluated using the method:

```
valueOfBonds
```

within the DurationConvexity EJB Component, which yields the values:

```
The weight of the 0 bond is 0.01818181818181818
The weight of the 1 bond is 0.03636363636363636
The weight of the 2 bond is 0.05454545454545454
The weight of the 3 bond is 0.07272727272727272
The weight of the 4 bond is 0.09090909090909091
The weight of the 5 bond is 0.10909090909090909
The weight of the 6 bond is 0.12727272727272726
The weight of the 7 bond is 0.14545454545454545
The weight of the 8 bond is 0.16363636363636364
The weight of the 9 bond is 0.18181818181818182
```

Duration of a Portfolio

Problem: What is the duration of the portfolio with assets of weight (0.25, 0.50, 0.25) and corresponding durations (0.73, 1.52, 3.34)?

Solution: The duration of the portfolio is 1.77749..., which can be evaluated using the method:

```
durationOfPortfolio (double[] duration, double[] weightOfBond)
```

from the `DurationConvexity` EJB Component. Which agrees with the solution in Example 6.

Duration of a Portfolio's Value

Problem: What is the duration of the portfolio of US T-Bonds which consists of three holdings with a market value a \$250,000, \$500,000 and \$250,000; with corresponding durations of 0.73, 0.152 and 3.34?

Solution: The duration of the portfolio is 1.77749..., which can be evaluated using the method:

```
durationOfPortfolio (double[] duration, double[] weightOfBond)
```

from the `DurationConvexity` EJB Component. Which agrees with the solution in Example 5.

Percentage Price change of Portfolio

Problem: What are the percentage price changes of a portfolio which contains the following assets:

Asset 1	duration = 2.05	valueOfBond = 50030
Asset 2	duration = 3.65	valueOfBond = 25802
Asset 3	duration = 1.65	valueOfBond = 15620

and experiences a parallel shifts in the yield curve of 0.005, 0.01, 0.02 (i.e. 0.5%, 1% and 2%)?

Solution: The percentage price change can be evaluated using the method:

```
percentagePriceChange (double deltaYield, double[] duration, double[] valueOfBond)
```

within the `DurationConvexity` EJB Component, which yield the price changes of:

- When the yield curve experiences a shift of 0.005 then the change in price of the portfolio is 0.012165496653982415, or equivalently 1.2165496653982415%
- When the yield curve experiences a shift of 0.01 then the change in price of the portfolio is 0.02433099330796483, or equivalently 2.433099330796483%
- When the yield curve experiences a shift of 0.02 then the change in price of the portfolio is 0.04866198661592966, or equivalently 4.866198661592966%

Percentage Price change of a Bond portfolio

Problem: What is the percentage price change of the bond portfolio with a duration of 2.25 and a yield of 4%, which experiences a parallel shift in the yield curve of 0.01 (i.e. 1%)?

Solution: The percentage price change is evaluated using the method:

```
percentagePriceChange (duration, annualYield, deltaYield)
```

within the `DurationConvexity` EJB Component, giving a percentage change of $-0.0216\dots$, or equivalently $-2.16\dots\%$.

Percentage Price change of a bond portfolio (semi-annual)

Problem: Using the same situation as in the previous example, except that here the yield of 4% is expressed semiannually (i.e. equivalent annual yield is less than 4%) and we wish to know the percentage price change?

Solution: The percentage price change is evaluated using the method:

```
percentagePriceChange (duration, yield, periodsCompoundedOver, deltaYield)
```

within the `DurationConvexity` EJB Component, giving a percentage change of $-0.0220\dots$, or equivalently $-2.20\dots\%$.

Duration in a General Setting

Problem: What is the duration of an futures contract on a zero coupon bond which has a market price of \$10 and is known to experience a \$0.50 price change for a 1% change in the zero coupon yield curve?

Solution: The duration is evaluated using the method:

```
durationGeneral (price, deltaZero, changeInPrice)
```

within the `DurationConvexity` EJB Component, giving a value of the duration of -0.5 .

Price change in the General Setting

Problem: The percentage change the price of the above futures contract under a change in the zero curve of 2% is?

Solution: The price change is evaluated using the method:

```
percentagePriceChangeGeneral (duration, deltaZero)
```

within the `DurationConvexity` EJB Component, giving a percentage price change of 0.1, or equivalently 10%.

Convexity of a Bond Portfolio

Problem: What is the convexity of a bond portfolio which consists of one bond which pays an annual coupon of 1, with the next payment due in exactly 6 months and where the bond has exactly 5 years to maturity and yields 6%?

Solution: The convexity of the bond is evaluated using:

```
convexity (yield, payments, timeOfPayments)
```

from the `DurationConvexity` EJB Component, giving a result of 8.1419....

Convexity when the Price is known

Problem: What is the convexity of a bond portfolio which consists of one bond which pays an annual coupon of 1 and has a market price of \$90? The next payment is due in exactly 6 months and the bond has 5 years to maturity.

Solution: The convexity of the bond when its market price is known is evaluated using:

```
convexityFromPrice (price, coupons, timeOfCoupons)
```

from the `DurationConvexity` EJB Component, giving a result of 0.36547....

Chance in Price taken Convexity into account

Problem: What is the percentage change in the price of a portfolio of UK Gilts with a duration 4.1 years and a convexity of 0.43, when the yield of the portfolio changes by 25 basis points?

Solution: The price change taking the convexity into account is given by:

```
bondPriceChange (duration, convexity, yieldChange)
```

from the `DurationConvexity` EJB Component, giving a result of $-0.01024\dots$, or equivalently -1.02% .

Applying the Duration Hedge Ratio

Problem: An investor wishes to hedge his exposure from a bond portfolio to interest rate risks by the use of interest rate futures over the next 3 months. The investor holds a portfolio of T-bonds with an estimated forward duration of 5.3 years and a forward value of \$1,315,040, at the end of the intended hedge in 3 months.

The current price of the T-bond future is 94 and each futures contract represents an underlying quantity of \$100,000 face value of bonds. Therefore, the futures contract price is \$94,000. The expected cheapest-to-deliver bond at the expiry of the futures contract has a duration of 8.20 years. In order to hedge the T-bond portfolio the investor needs to sell T-bond futures. What is the optimal number of T-bond futures which the investor will need to sell in order to provide the best hedge during the next 3 months?

Solution: The optimal number of T-Bond future the investor will need to sell in order to hedge the T-bond portfolio is 9. Since this is the closest number of contracts to the Duration hedge ratio which is 9.042..., which is evaluated using the method:

```
durationHedgeRatio(94000, 8.20, 1315040, 5.3)
```

from the DurationConvexity EJB Component.

7.3.5 Database Example with JDBC Mediator

The Database Example is located inside the *Client/BasicBonds/DatabaseExample* 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 a table named COUPONS. For example, if you have decided using the 'test' database, you would optionally 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.mysql.jdbc.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
```

7.4 Yield of Fixed-Interest Bonds on Interest Payment Dates

Within this section we illustrate the use of the methods provided by the `FixedInterestBonds` EJB Component. The source code for these examples can be found within the:

`FixedInterestBondsClient`

console client which is contained within the **Client** folder of this installation package.

7.4.1 Gross Interest Yield

Problem: Let’s assume that the quoted price of 10% bond is £102 (per £100 nominal). We need to calculate the interest yield for an investor who pays no tax.

Solution: The gross interest yield is 0.098, which can be calculated using the:

```
grossInteretYield(0.10, 102)
```

method within the `FixedInterestBonds` EJB Component.

7.4.2 Gross Redemption Yield

Problem: A treasury bond has a term to redemption of exactly 14 years. If the coupon rate is 13% and the current market price is 123.9 what is the gross redemption yield?

Solution: The gross redemption yield is 0.098, and can be evaluated using the:

```
grossRedemptionYield(123.0,13,14)
```

method within the `FixedInterestBonds` EJB Component.

7.4.3 Holding Period Return

Problem: An investor who pays no tax purchases a treasury bond with coupon rate 12.5% at a price of \$114.25. Accrued interest on the settlement date was zero. After holding the bond exactly 3 years, the investor sold at \$126.75. Ignoring expenses, what is the holding period return per half-year?

Solution: The holding period return is 0.069, and can be evaluated using the:

```
holdingPeriodReturn (114.25, 12.5, 126.75, 3)
```

method within the `FixedInterestBonds` EJB Component.

7.5 Interest Investments

7.5.1 Compound Interest

Problem: If Charles deposits \$100 in a savings account at a federally insured bank now at 12% interest, how much will accumulate in the account in 7 years if the interest accumulates annually? How much will Charles accumulate if interest is compounded semiannually?

Solution: Charles will accumulate \$221.06 with annual compounding and \$226.09 with semiannual compounding.

Source Code: The **source code** for this example can be found in *Client/Interest/CompoundInterest* folder.

7.5.2 Effective Rate of Interest

Problem: You have an amount of money and you want to put them into a bank so it will earn interest. You have two offers:

- make an annual deposit with 4% yearly interest rate
- make a semiannual deposit with 2.5% semiannual interest rate

Knowing that you want to keep the money at the bank for a year, which is the best offer?

Solution: One way to solve this problem is to convert the annual rate of 4% into semi-annual rates. To perform this task using the method **convertYearlyRate** from the **EffectiveAndNominalInterests** EJB Component.

Source Code: The **source code** for this example can be found in *Client/Interest/EffectiveAndNominalInterests* folder.

7.5.3 The accumulated value

Problem: Calculate the value of a series of payments of one per interval, payable in arrears for 5 intervals when the interest is 6%?

Solution: To calculate this we will use the method **accumulatedSeriesOfPayments** method from the **AccumulatedValuesOfAnnuityCertain** EJB Component.

Source Code: The **Java source code** for this example can be found in *Client/Interest/AccumulatedValue* folder.

7.5.4 Present Values

Problem: You need to decide between two business opportunities. The first opportunity will pay \$700 in 4 year's time and the second opportunity will pay \$850 in 6 year's time. What represents the best business opportunity when the interest rate is expected to average at 8% annually over the next four and six years?

Solution: We calculate the discount of these future sums and hence calculate the initial amount of money that has to be invested in order to receive the respective future sum. To do this we will use the **presentValue** method from the **PresentValues** EJB Component. The results will give:

- \$514.52 for the first opportunity
- \$535.64 for the second opportunity

On the basis of the present value calculation we would choose the second opportunity since it represents an opportunity with a higher present value.

Source Code: The **source code** for this example can be found in *Client/Interest/PresentValue* folder.

7.5.5 Yield

Problem: What is the yield of an investment over six years if the investment makes an interest payment in the first, second, third and sixth years at a rate of 5%?

Solution: To solve this problem one should use the method **yield** from the **ValueReturnYield** EJB Component. Set the array argument to [**true true true false false true**], the number of periods to 6 and the interest to 0.05.

Source Code: The **source code** for this example can be found in *Client/Interest/Yield* folder.

7.5.6 Repo agreements

Problem: What is the real return from an investment of \$100.000, which increases in value to \$150.000, after one year where the rate of inflation was 1%?

Solution: To solve this problem one should use the method **realReturn** from the **RealReturns** EJB Component.

Source Code: The **source code** for this example can be found in *Client/Interest/Repo* folder.

Chapter 8

Bonds UML Diagrams

8.1 Introduction and Key

Within this chapter we provide UML diagram(s) for the interfaces of the Bonds component¹. The diagrams use standard UML notation detailing the inheritance, dependency, association, interface, methods, fields and properties of this component and associated packages.

UML Diagram Key

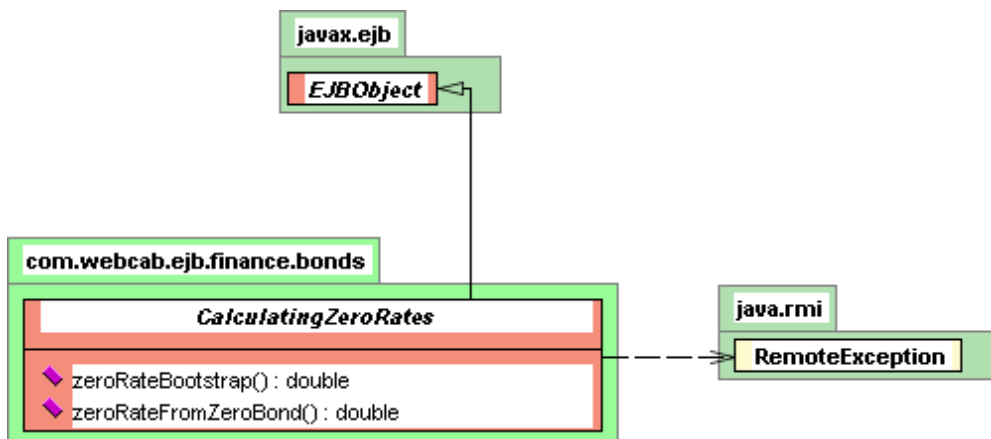
- **Extended Classes (interfaces)** - A solid line with a large triangle that points from the subclass (resp subinterface) to the superclass (resp inherited interface)
- **Classes** - Displayed in a rectangular box with a default yellow background with the name at the top and field, methods, and properties listed below it
- **Implementing Classes** - A dashed line with a large triangle which points from the implementing class to the inherited interface
- **Interfaces** - A rectangle with orange background and the interface name in italic font
- **Implemented Interfaces** - A dashed line with a large triangle
- **Dependencies/Associations** - A dashed/solid line with an arrowhead
- **Packages** - A green rectangle with a tab and the package name in the tab or below it
- **Methods** - Listed below members and fields, including the return type
- **Members/fields** - Listed below the class name, including the return type
- **Static** - Static members, fields, variables, and methods are underlined in the diagram

¹We present the UML diagrams with a large amount of white space so that the developer has space in which to add additional remarks.

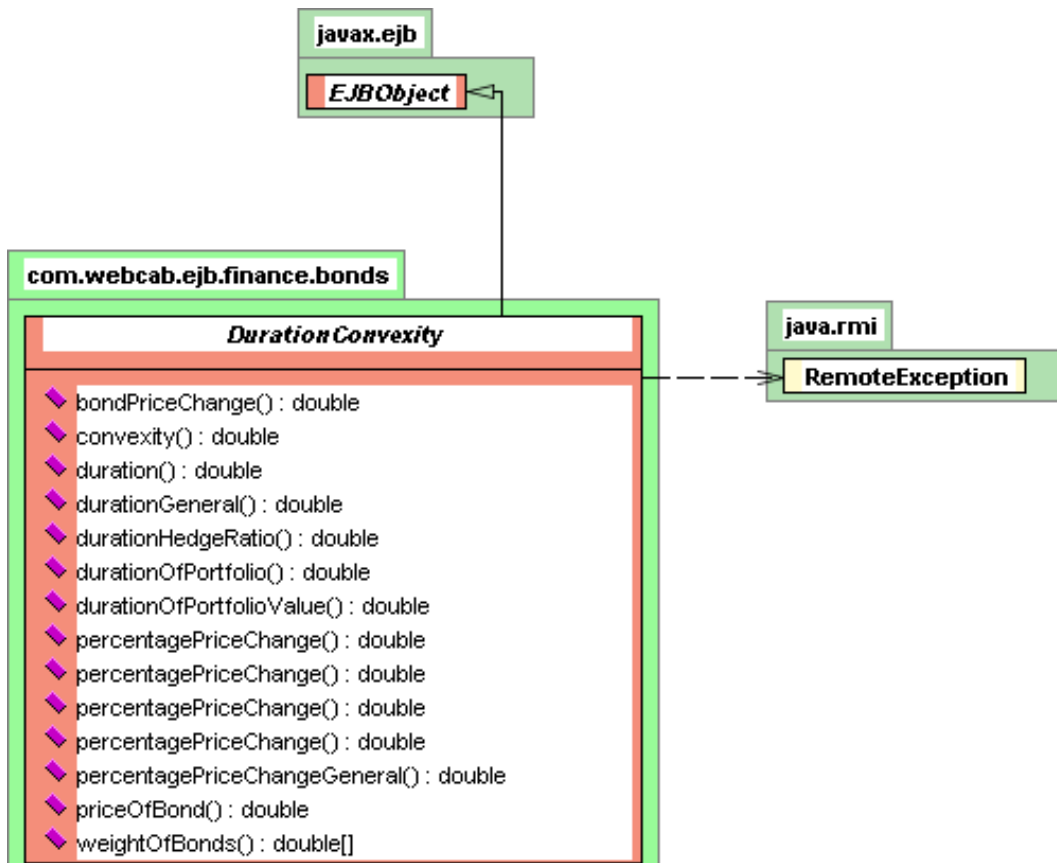
8.2 UML Diagrams

Zoom in order to be able to read the names of the fields, methods, components and packages.

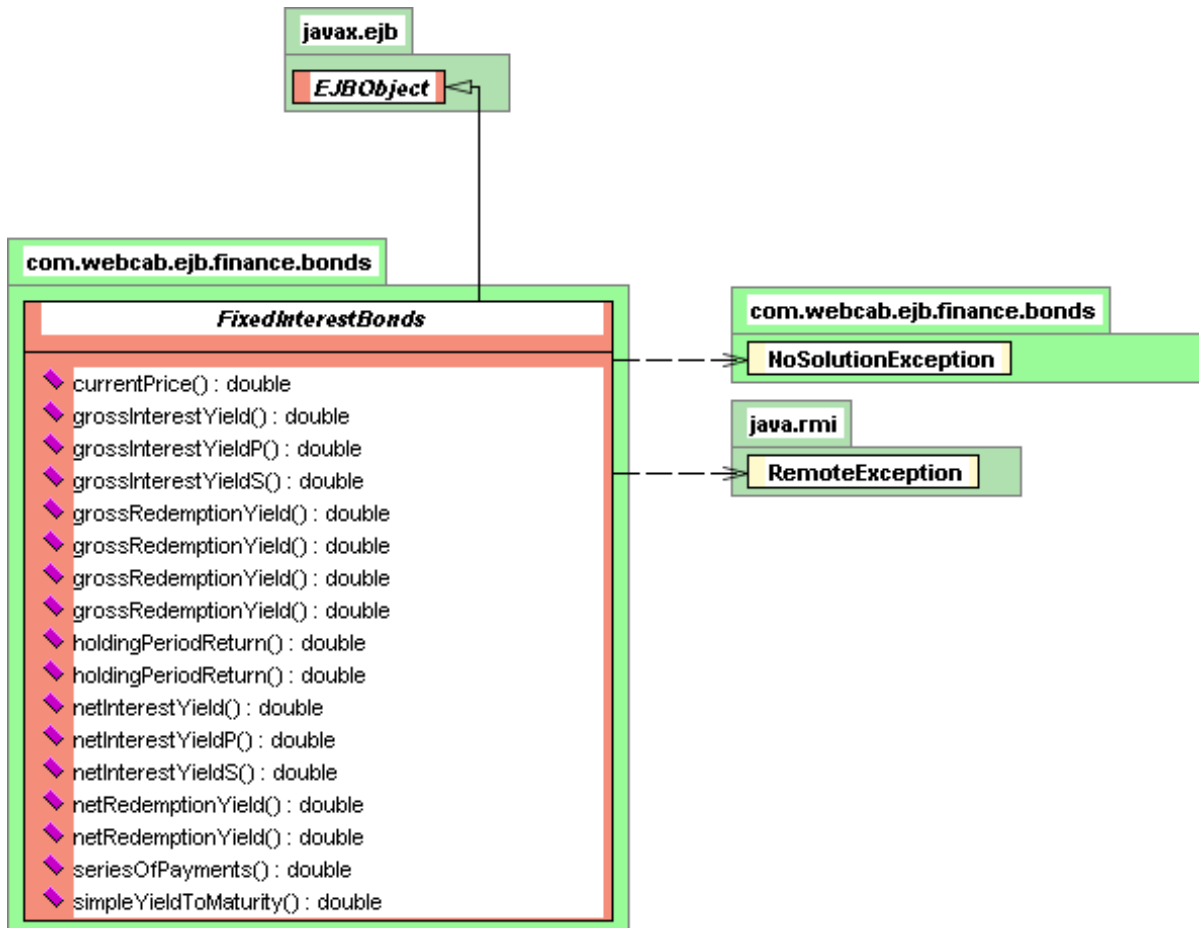
8.2.1 CalculatingZeroRates



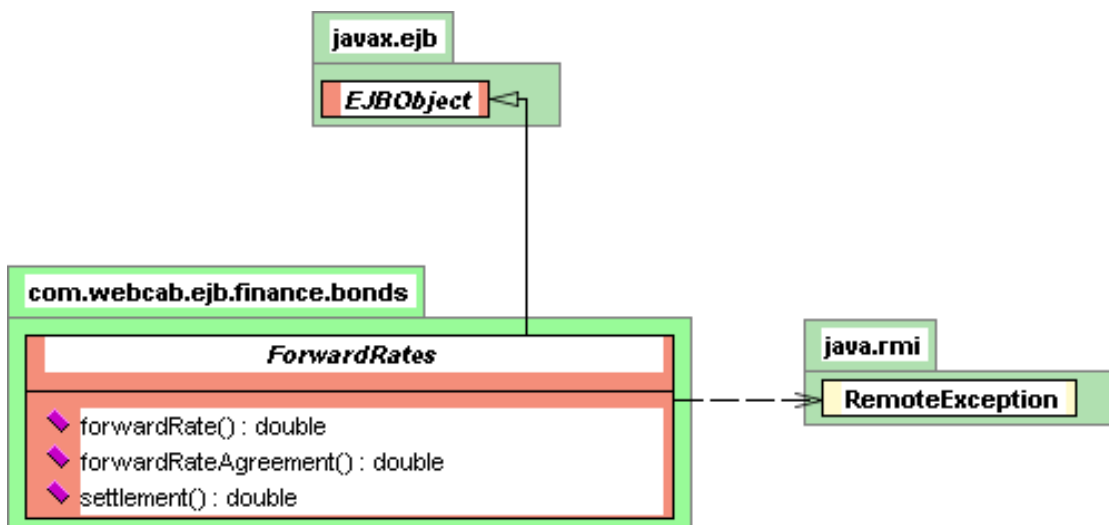
8.2.2 DurationConvexity



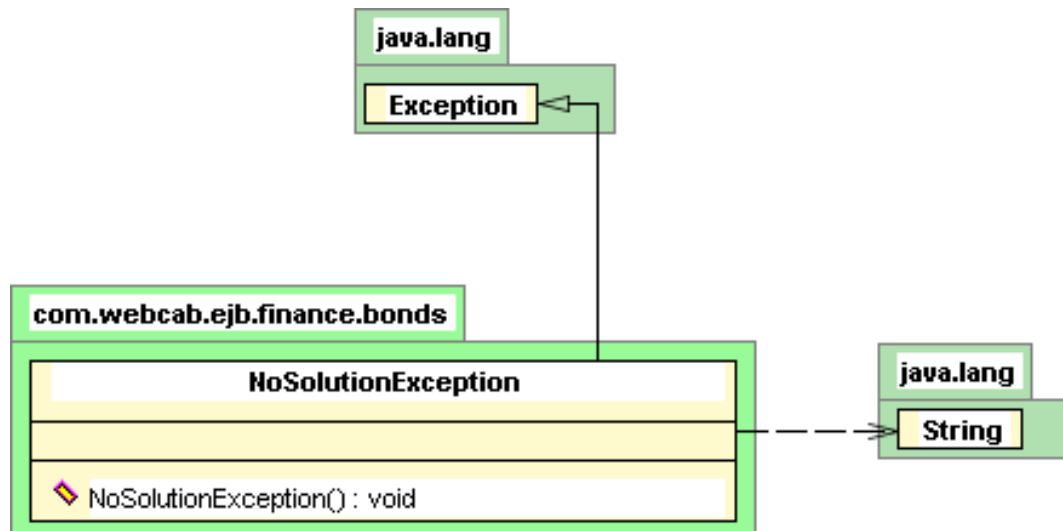
8.2.3 FixedInterestBonds



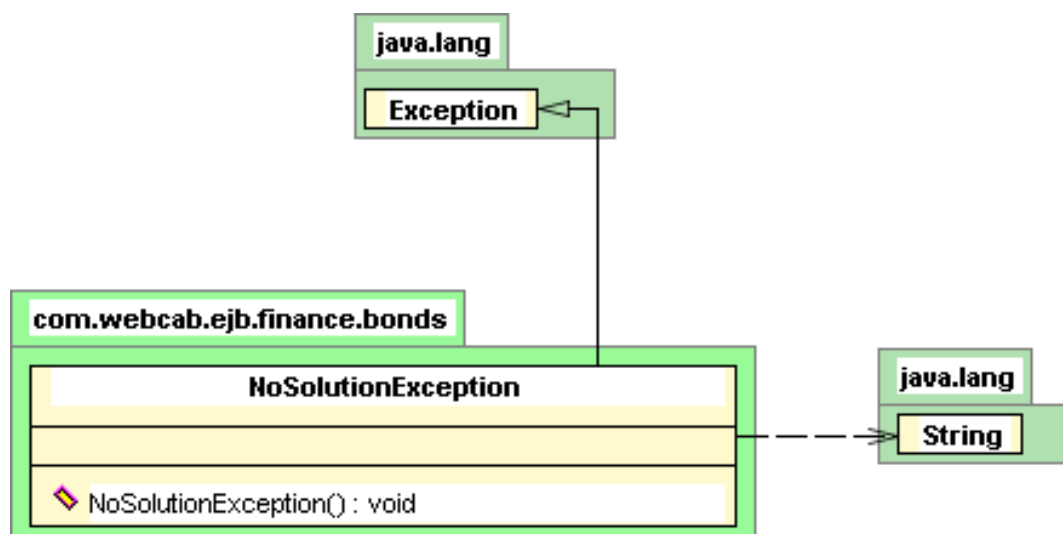
8.2.4 ForwardRates



8.2.5 NoSolutionException



8.2.6 TreasuryPrices



Chapter 9

Guide to WebCab Components

9.1 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 on the J2SE, J2EE and .NET platforms.

9.2 Presentation of Products

We aim to provide good quality, useful information to help you decide which J2EE application 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.

9.3 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 enterprise applications 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 J2EE Applications with the following technologies:

- Client containers - Internet Explorer, Netscape Navigator, Opera
- IDEs - JBuilder, Eclipse, BEA Studio, Sun ONE (formerly Forte For Java), IBM VisualAge, Oracle JDeveloper
- Client Side Technologies - Application Clients, Servlet, JSP, Applets
- EJB containers - IBM WebSphere, BEA WebLogic, Oracle 9iAS, Sun ONE AppServer, Borland AppServer, Sybase EAServer, Ironflare Orion, JBoss

- DBMS - Oracle, IBM DB2, SQL Server, Sybase, MySQL
- Platforms - Windows XP/2000/NT, Linux, Solaris, IBM AIX

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

9.4 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.

9.5 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.

9.6 Company Culture and Activity

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

9.7 Product Life cycle

We continuously add value to our products by evolving them according to new J2EE specifications, customer feedback and market demands. We give particular emphasis to incorporating our clients suggested modifications and additions into our product development cycle.

9.8 Support, Warranty and Upgrades

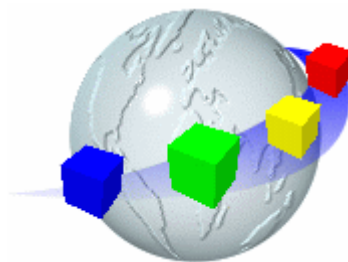
WebCab warrants that each component will perform substantially in accordance with the accompanying written material. We provide with all our products without charge sixty (60) days product support services including fixing bugs, compatibility issues and other technical support issues.

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. .NET is a trademark of Microsoft Corporation in the U.S. and other countries