Title	PortfolioXS Bean Documentation (PortfolioXS.doc)
Version	0.1 (draft)
Author	S.H.P. Janszen <bas@marketxs.com>
Distribution	-
Date	04/07/00

marketXS

1 PortfolioXS description

PortfolioXS enables your users to obtain a current report of their personal investment portfolio and investment results at any given moment. One or more share portfolios can be built up and closely monitored in an easy way, making purchasing and selling decisions easier and faster. Simplified Watch lists can be produced for a quick and global overview of a number of selected stocks.

PortfolioXS works based on a clear menu structure and provides the user with a step-by-step guide for building up and/or changing their investment portfolio(s). A log-in procedure ensures that the data can only be consulted in a secure environment. This log-in procedure can be disconnected if required.

Each user can build up and monitor 10 different portfolios simultaneously. For each portfolio, 10 different shares from various stock exchanges can be selected. Your users determine themselves which information is to be displayed when they compile their portfolios. PortfolioXS offers a choice of a large number of different variables divided into three categories, 1) price and market information, 2) financial ratios and fundamental information, 3) the personal investment results of the user:

Price and market information:

Latest price, price change (%), bid and asked prices, highest and lowest price of the day, highest and lowest annual price, transaction volume, average volume, index and change in index (%). The module's standard setting uses delayed-price information.

Financial ratios and fundamental information:

Price/earnings ratio, market capitalization, price/turnover ratio, etc.

Personal investment results:

Purchase price, number of shares, purchase date, purchase value of shares, current value of shares, change in value (%) and total value of portfolio. This category is to be expanded to include development of cash flow.

An 'alert' system is being added to PortfolioXS: a system warning the user on matters concerning important events (news releases), sharp changes in price and/or whether specific financial ratios of shares in the portfolios have been exceeded. Here again, it is the user who determines which variables are included in the alert system. The alert system is also designed with a clear menu structure and provides the user with a step-by-step guide for determining the variables.

The *Watch list* module provides your users with a quick and simplified survey of the market development of a number of selected shares. As well as the ticker code, the Watch list shows the latest price and the price change (%) in relation to the previous closing price. In each Watch list, 10 different shares can be monitored simultaneously, and again this module is designed using a clear menu structure.

The PortfolioXS classes

The **PorfolioXS** module works together in combination with the **QuoteXS** module. The **QuoteXS** classes reside in the package com.marketxs.quote. The **PortfolioXS** classes reside in the com.marketxs.com.profile package. Make sure to import these packages in your ***servlet***. Other required packages are:

```
javax.servlet.*;
javax.servlet.http.*;
javax.naming.*;
javax.ejb.*;
java.rmi.*;
javax.transaction.*;
```

The following table lists the classes that you get with the **PortfolioXS** package.

Class Name	**Type**	**Description**
ProfileHome	***JNDI*** (home) interface	This class is used to get a reference to the Profile object.
Profile	Remote Interface	Used to add updates and deletes users from the MarketXS user database.
ProfileData	Data class	Contains the actual user information (profile)
PortfolioHome	***JNDI*** (home) interface	This class is used to get a reference to the Portfolio object.
Portfolio	Remote Interface	Retrieves, adds, updates and deletes portfolios from the MarketXS user database.
PortfolioData	Data class	The PortfolioData class describes the user's portfolio object.
PortfolioContentHome	***JNDI*** (home) interface	This class is used to get a reference to the PortfolioContent object.
PortfolioContent	Remote Interface	
PortfolioContentData	Data class	This class represents a portfolio entry (stock/share) in the user's portfolio.
WatchlistHome	***JNDI*** (home) interface	This class is used to get a reference to the Watchlist object.
Watchlist	Remote Interface	Retrieves, adds, updates and deletes watchlists from the user database
WatchlistData		The Watchlist class describes the user's watchlist.
WatchlistContentHome		
WatchlistContent	Data class	
WatchlistContentData		
QuoteHome	***JNDI*** (home) interface	This class is used to get a reference to the Quote object.
Quote	Remote Interface	This class is used to retrieve quote information. It returns QuoteData objects.
QuoteData	Class	QuoteData objects contain the stock exchange information fields for a ticker or symbol.
LookupHome	***JNDI*** (home) interface	
Lookup	Remote Interface	Utility class used to find tickerIds (primary key for) quotes based on either the symbol or the company name
TickerData	Data class	Holds ticker information (subset of QuoteData fields)

1.1 The ProfileHome, Profile and ProfileData classes

The **Profile** class is used to communicate with the **ProfileData** class, which holds user information. A **ProfileData** object is a required parameter to get access to *real-time* quotes. At present, this option is only available in combination with our **PortfolioXS** product.

Using the profile class the reseller has access to the following functionalities:

- Add users
- Update users
- Delete users
- Retrieve detailed information about a particular user
- Check if a particular user exists
- Get a user's quota (restrictions to the maximum allowed number of *real-time* quotes)
- Retrieve reseller information

Example of updating a user in the database (assuming you have done the required *imports for* **servlets**, **JNDI** *and* ***RMI***)

```
// First we connect to the JNDI service and log in to the application server.
Hashtable hash = new HashTable();
try
{
 hash.put(Context.INITIAL_CONTEXT_FACTORY,"weblogic.jndi.WLInitialContextfactory");
 hash.put(Context.PROVIDER_URL,"t3://jndi.foobar.com:7001");
 hash.put(SECURITY_PRINCIPALS,"username");
 hash.put(SECURITY_CREDENTIALS."password");
}
catch(NamingException e) {}
Context ctx = new InitialContext(hash);

// Now we're ready to look up the bean's home Interface, get a reference ph to it and use that to retrieve
// a reference to a Profile object. The Profile object can then be used to get a user's profile (ProfileData) with
// its getProfile() method. The ProfileData object holds all the user information,which can be changed and
// updated in the database.
try
{
 ProfileHome ph = (ProfileHome)ctx.lookup("com.marketxs.profile.Profile");
 Profile profileManager = ph.create();
 ProfileData userProfile = profileManager.getProfile("jdoe");

 if (userProfile.isOwner(userProfile))
 {
  userProfile.setLastname("Doe");
  profileManager.update(userProfile);
 }
else
 {
  System.out.println("You do not own this profile.");
 }
}
catch(RemoteException e){System.out.println("RemoteException oocured");}
catch(NoSuchUserException e) {System.out.println("NoSuchUserException occured);
catch(CreateException e) {System.out.println("CreateException occured");}
```

For a list of all available methods please see the API.

1.2 The PortfolioHome, Portfolio and PortfolioData classes

The PortfolioData object describes the characteristics of a portfolio of a user.

The PortfolioHome class is used to get a reference to the Portfolio object. The Portfolio object manages PortfolioData objects. Via the Portfolio class PortfolioData objects may be added, deleted and updated from a user's portfolio.

A user is represented in the database by a Profile object. The PortfolioData objects represent a portfolio that belongs to that user profile. A single user may have multiple portfolios. Therefore, many PortfolioData objects may exist for a single profile (user).

The Portfolio class will let you retrieve the actual portfolio's information by supplying it either a ProfileData object or a PortfolioId. The returned PortfolioData object contains portfolio information like the name and description of the portfolio, currency, ordertype, PortfolioId, rank and the properties to display. See the API for a full listing of methods.

A portfolio on its turn may consist from multiple company symbols or indices that are of interest to the user. The PortfolioContentData object represents an item on the user's portfolio.

TODO EXAMPLE

1.3 The PortfolioContentHome, PortfolioContent and PortfolioContentData classes

A PortfolioContentData object represents each item in the user's portfolio.

The PortfolioContentHome class is used to get a reference to the PortfolioContent object. The PortfolioContent object manages PortfolioContentData objects. Via the PortfolioContent class PortfolioContentData objects may be added, deleted and updated from a user's portfolio.

PortfolioContentData objects contains the following:

- A PortfolioContentId (key)
- A PortfolioId(foreign key). Tells you to which portfolio this item belongs to.
- A description of the portfolio's entry
- The purchase date
- The purchase price
- The rank
- The share
- The TickerId of the item of interest.

All this fields may be retrieved from existing items or set for new PortfolioContentData objects. The newly created objects may be added to a user's portfolio via the addTo(PortfolioData) method.

TODO EXAMPLE

1.4 The WatchListHome, WatchList and WatchListcontent classes

TODO EXAMPLE

A user may have entries in his portfolio and a watchlist. A watchlist is one of more quotes of interest to the end user. A watchlist is comparable to a portfolio with the difference that a watchlist entry contains less information than a portfolio entry (i.e. a watchlist is a subset of a portfolio).

The Watchlist class is a class used to:

- add new watchlist's to a user's profile . A user may create more than one watchlists
- editing watchlist
- remove watchlist's from a user's profile

The WatchlistData object represents one of the user’s watchlists.

Method Summary	
`WatchlistData`	`add(WatchlistData watchlistData)` Adds a new watchlist.
`void`	`delete(WatchlistData watchlistData)` Remove an existing watchlist.
`boolean`	`exists(WatchlistData watchlistData)` Use this method as an inexpensive way to verify the existence of a watchlist. It returns `true` if this reseller has a watchlist with the same id as the provided ProfileData, `false` if not. This method is lightweight.
`WatchlistData`	`getWatchlist(long watchlistId)` Returns a watchlist.
`java.util.Collection`	`getWatchlists(ProfileData profileData)` Returns all watchlist's of the given user's profile.
`WatchlistData`	`update(WatchlistData watchlistData)` Update the information in a watchlist. This does not include the watchlist entries such as tickers, only its name, ranking etc.

1.5 The QuoteHome, Quote and QuoteData classes

A QuoteData object represents real-time or delayed quote information. QuoteData objects are retrieved through the Quote object. The QuoteHome class is used to get a reference to the Quote object. The Quote objects returns QuoteData objects or collections of QuoteData objects depending on the input you give it (a single tickerid or a collection of tickerIds). Whether the returned QuoteData object is real-time data or delayed data depends on whether you also supply a user’s ProfileData object or not to the Quote object.

Example:

```
…

try
{
 Context ctx = getInitialContext();
 QuoteHome qh = (QuoteHome)ctx.lookup("com.marketxs.quote.Quote");
 q = qh.create();
 QuoteData qd = getQuote(q, tickerId, mask);

 out.println("Last quote for tickerId "+tickerId+" = "+qd.getLast());
}
catch (NamingException e){out.println("Ooops: NamingException occured!");}
catch (CreateException e){out.println("Ooops: CreateException occured!");}

…

public QuoteData getQuote(Quote q, long tickerId, String mask)
{
 QuoteData quoteData = new QuoteData();

 try
 {
  quoteData = q.getQuote(tickerId, mask);
 }
 catch(RemoteException e){}
 catch(QuoteException e) {}

 return quoteData;
}
```

As you can see in the example, supplying the Quote object with a tickerId retrieves QuoteData objects. A tickerId is anything that holds a value in the MarektXs database. This could be a share or option of a public company or an exchange index and so on. To look up the tickerId, which is required for the Quote object, a utility class Lookup exists. This class allows you to retrieve tickerIds by company name or symbol.

In addition, a mask string is provided to the getQuote method. This mask identifies which fields to return for the QuoteData object. The string is composed from zeroes and ones, which represent whether to return the field or not. Currently, there are 56 fields. For example, to retrieve the High and Low values for a quote the string
“0000000000000000000000000000000000000000000000110000”
is passed on as the mask parameter.

For a complete listing of all the available fields, please see appendix A.

1.6 The LookupHome, Lookup and TickerData classes

The **Lookup** class is a class that is used to lookup tickerIds. A tickerId is the internal key we use to uniquely identify a company's quote for a specific stock exchange. All of our objects that query our database require the tickerId as a parameter. For example, the **Quote** class is passed a tickerId to construct and return a **QuoteData** object that is filled with information about this tickerId (quote). To find the tickerId for a quote we provide you with a class that returns the tickerId(s) based on either the given stock symbol or the company name. If the company is listed on one stock exchange, a Collection is returned containing a single element. If the company is listed on more than one stock exchange, the returned Collection is filled with tickerIds for every stock exchange the company is listed on. If no listings are found for this company, the class throws an exception.

See the API for a complete listing of all the method for this class.

Example:

```
long tickerId;
String symbol;

res.setContentType("text/html");
out = res.getWriter();


// retrieve te symbol from the HTTP request, use AAB as the default if none is supplied.
symbol = req.getParameter("symbol") == null ? "AAB" : new String(req.getParameter("symbol"));

try
 {
  Context ctx = getInitialContext();
  LookupHome slh =(LookupHome)ctx.lookup("com.marketxs.quote.Lookup");
  Lookup sl  = slh.create();

  QuoteHome qh = (QuoteHome)ctx.lookup("com.marketxs.quote.Quote");
  Quote q = qh.create();

  Vector tickerVector = new Vector(sl.getTickersBySymbol(symbol, 1));
  Enumeration tics = tickerVector.elements();

  while (tics.hasMoreElements())
  {
   TickerData tic = (TickerData) tics.nextElement();
   tickerId = tic.getTickerId();
   ShortQuote sq = (ShortQuote) q.getQuote(tickerId, new ShortQuote());
   out.println("Last quote for tickerId "+tickerId+" = "+sq.getLast()+" ("+sq.getName()+"
"+sq.getMarketName()+")<BR>");
  }
 }
 catch (NamingException e){out.println("NamingException occured!");}
 catch (CreateException e){out.println("CreateException occured!");}
 catch (LookupException e) {out.println("LookupException occured!");}
 catch (QuoteException e) {out.println("QuoteException occured!");}
….

The TickerId object holds the following fields:

TickerId (unique key)
Symbol (unique alphanumeric code for this ticker)
MarketId (unique key for the exchange of origin)
MarketAbbrev (the abbreviation for the exchange of origin)
marketName (the name of the exchange of origin)
Name (name of the company/fund)
```

2 Servlet example

This documentation assumes you are using a servlet to connect to the **MarketXS** modules. This chapter describes how the servlet gets access to the accessor methods as implemented in the **QuoteXS** module. The **QuoteXS** module provides access to all quote data, real time and or delayed. In order to access the methods that the bean provides for, you must first create the servlet itself. The standard 1.2 ***JDK*** from **Sun Microsystems** may be used for this.

You can download Sun's ***JDK*** from **Sun**'s Java website (http://www.javasoft.com). You must make a class that extends the HttpServer class. You will need to import the Java Servlet class files (http://java.sun.com/products/servlet/download.html). As your servlet will connect to **Enterprise Java Beans (EJBs)** located on **MarketXS'** application server via **Java *RMI*,** you will also need the ***EJB*** class files (/http://java.sun.com/products/ejb/docs.html) as well as the ***JNDI*** class files.

(Note: your ***JDK*** may or may already have the ***JNDI*** classes. If not, you may download them at http://java.sun.com/products/jndi/)

With these classes in your $CLASSPATH (Servlet classes, ***EJB*** classes and the ***JNDI*** classes) you are almost ready to code your first servlet. You need to install the **MarketXS** package, which hold the stubs for the beans. You should have received a ***jar*** file from **MarketXS**, which contains this package. Have your $CLASSPATH point to our ***jar*** file and import the following in your servlet:

```
// required for the servlet
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// required for EJ Beans, RMI remote objects and JNDI lookups.
import javax.ejb.*;
import java.rmi.*;
import javax.naming.*;
import java.util.*;

// The MarketXS.com package for the RMI stubs for the QuoteXS objects
import com.marketxs.quote.*;
```

Now you have all the necessary components to build a connection to our **QuoteXS** bean and use it method to retrieve quote information. The first thing you should do in your servlet is to log on our application server using your company's username and password. In order to do that you need to set up the context for the connection by setting some environment properties. Creating an instance of he javax.naming.InitialContext (which implements the javax.naming.Context interface) and setting some properties by passing it a hash does this for you. You should set the following four properties.

Property name	**Description**	**Example**
INITIAL_CONTEXT_FACTORY	Weblogic Application Server	"weblogic..jndi.WLInitialcontextFactory"
PROVIDER_URL	The URL to the application server (protocol://machine:port"	"t3://some.machine.com:6001"
SECURITY_PRINICIPALS	Username for the application server	"foo"
SECURITY_CREDENTIALS	Password for the application server	"bar"

You can see how these properties are set in the sample code for the servlet provided in this document.

Once you have set these properties and the instance of the InitialContext object has been created you are logged on to the WebLogic application server. The second step is then to lookup the bindings for the **QuoteHome** object. This is achieved by calling the lookup method of the Context object we now have a reference to. This method should be supplied with a string, which represents the path to the Quote bean ("com.marketxs.quote.Quote") on the application server. The method returns an object that should be type casted to an object of type QuoteHome. Now that we have a QuoteHome object we are ready to create a reference to a Quote object using the QuoteHome's create() method. Once we have this object we can create a **QuoteData** object with it. This is what we've been after all the time!

3 Glossary

API - Application **P**rogrammer's **I**nterface.
An API (application program interface) is the specific method prescribed by a computer operating system or by another application program by which a programmer writing an application program can make requests of the operating system or another application.

CORBA - Common **O**bject **R**equest **B**roker **A**rchitecture
CORBA is the acronym for **C**ommon **O**bject **R**equest **B**roker **A**rchitecture. It is an open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can inter-operate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.

DNS - Domain **N**ame **S**ervice
A network service used to map domain names (e.g. marketxs.com) into IP addresses and vice versa.

EJB - Enterprise **J**ava **B**eans
Enterprise JavaBeans technology is a scalable, distributed and cross-platform architecture that lets developers tap into and utilize enterprise-class services. The technology allows developers to write business logic to the Java™ platform and easily integrate and leverage their existing enterprise investments.

HTML - Hyper**T**ext **M**arkup **L**anguage
HTML is the *language* used for publishing hypertext on the World Wide Web. It is a non-proprietary format based upon SGML, and can be created and processed by a wide range of tools, from simple plain text editors - you type it in from scratch- to sophisticated WYSIWYG authoring tools. HTML uses tags such as <h1> and </h1> to structure text into headings, paragraphs, lists, hypertext links etc.

IP address
In the most widely installed level of the Internet Protocol (IP) today, an IP address is a 32-bit number that identifies each sender or receiver of information that is sent in packets across the Internet. When you request an HTML page or send e-mail, the Internet Protocol part of TCP/IP includes your IP address in the message. It will send the message to the IP address that is obtained by looking up the domain name in the URL you requested or in the e-mail address you're sending a note to. At the other end, the recipient can see the IP address of the Web page requestor or the e-mail sender and can respond by sending another message using the IP address it received.

JAR - **J**ava **AR**chive
JAR stands for **J**ava **AR**chive. It's a file format based on the popular ZIP file format and is used for aggregating many files into one. Although JAR can be used as a general archiving tool, the primary motivation for its development was so that Java applets and their requisite components (.class files, images and sounds) can be downloaded to a browser in a single HTTP transaction, rather than opening a new connection for each piece.

JDK - **J**ava **D**evelopment **K**it
The Java™ Development Kit (JDK™) contains the software and tools that developers need to compile, debug, and run applets and applications written using the Java programming language. The JDK software and documentation is free per the license agreement

JVM - Java **V**irtual **M**achine
The Java Virtual Machine is the cornerstone of Sun's Java programming language. It is the component of the Java technology responsible for Java's cross-platform delivery, the small size of its compiled code, and Java's ability to protect users from malicious programs. The Java Virtual Machine is an abstract computing machine. Like a real

computing machine, it has an instruction set and uses various memory areas. It is reasonably common to implement a programming language using a virtual machine; the best-known virtual machine may be the P-Code machine of UCSD Pascal.

JNDI- **J**ava **N**aming and **D**irectory **I**nterface.
The Java Naming and Directory Interface™ is a standard extension to the Java™ platform, providing Java technology-enabled applications with a unified interface to multiple naming and directory services in the enterprise. As part of the Java Enterprise API set, JNDI enables seamless connectivity to heterogeneous enterprise naming and directory services

ORB - **O**bject **R**equest **B**roker
In CORBA, an Object Request Broker is the programming that acts as a "broker" between a client request for a service from a distributed object or component and the completion of that request. Having ORB support in a network means that a client program can request a service without having to understand where the server is in a distributed network or exactly what the interface to the server program looks like. Components can find out about each other and exchange interface information as they are running.

OS - Operating **S**ystem
An operating system (sometimes abbreviated as "OS") is the program that, after being initially loaded into the computer by a bootstrap program, manages all the other programs in a computer. The other programs are called *applications*. The applications make use of the operating system by making requests for services through a defined *application program interface* (API). In addition, users can interact directly with the operating system through an interface such as a command language.

RMI – Remote **M**ethod **I**nvocation
RMI (Remote Method Invocation) is a way that a programmer, using the Java programming language and development environment, can write object-oriented programs in which objects on different computers can interact in a distributed network. RMI is the Java version of what is generally known as a remote procedure call (RPC), but with the ability to pass one or more objects along with the request. The object can include information that will change the service that is performed in the remote computer.

Servlet
Servlets are pieces of Java™ source code that add functionality to a web server in a manner similar to the way applets add functionality to a browser. Servlets are designed to support a request/response computing model that is commonly used in web servers. In a request/response model, a client sends a request message to a server and the server responds by sending back a reply message.

SMS - Short **M**essage **S**ervice
SMS is a service for sending messages of up to 160 characters to mobile phones that use Global System for Mobile (GSM) communication. GSM and SMS service is primarily available in Europe.

WAP - **W**ireless **A**pplication **P**rotocol
The WAP is a specification for a set of communication protocols to standardize the way that wireless devices, such as cellular telephones and radio transceivers, can be used for Internet access, including e-mail, the World Wide Web, newsgroups, and so on. While Internet access has been possible in the past, different manufacturers have used different technologies. In the future, devices and service systems that use WAP will be able to inter-operate.