marketXS

QuoteXS

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

1 Introduction

This document describes the QuoteXS module and its objects. It gives an overview of the classes that are available to the developer as well as some sample code.

2 QuoteXS module description

MarketXS' **QuoteXS** module provides in quote information about public companies. The bean gives a detailed overview of the most relevant financial information.

Latest quote
Time of last transaction
Volume latest transaction
Tick trend
Bid price (volume)
Asked price (volume)
Net price change (and%)
Highest and lowest quotation of the day
Highest and lowest annual price
Volume of the day
Average trading volume
Opening price
Closing price for previous day
Market

In addition, **QuoteXS** offers the following functionality to the application developer.

- *Symbol lookup*: search functionality for symbols and tickers.

- *Authorization* (real-time quotes only): An end-user has to be authorized before he can receive real-time quote information. The **QuoteXS** bean supports registration, authentication and authorization of end-users. There is no authorization required d*elayed* quote information.

In combination with other **MarketXS** modules, the **QuoteXS** bean functionality can be expanded with the following:

- *Portfolio management*: Add a share with a single-click to an end-user's portfolio (in combination with **PortfolioXS**).
- *Financials*: Get financial information about a company with a single-click (in combination with **FinancialXS**).

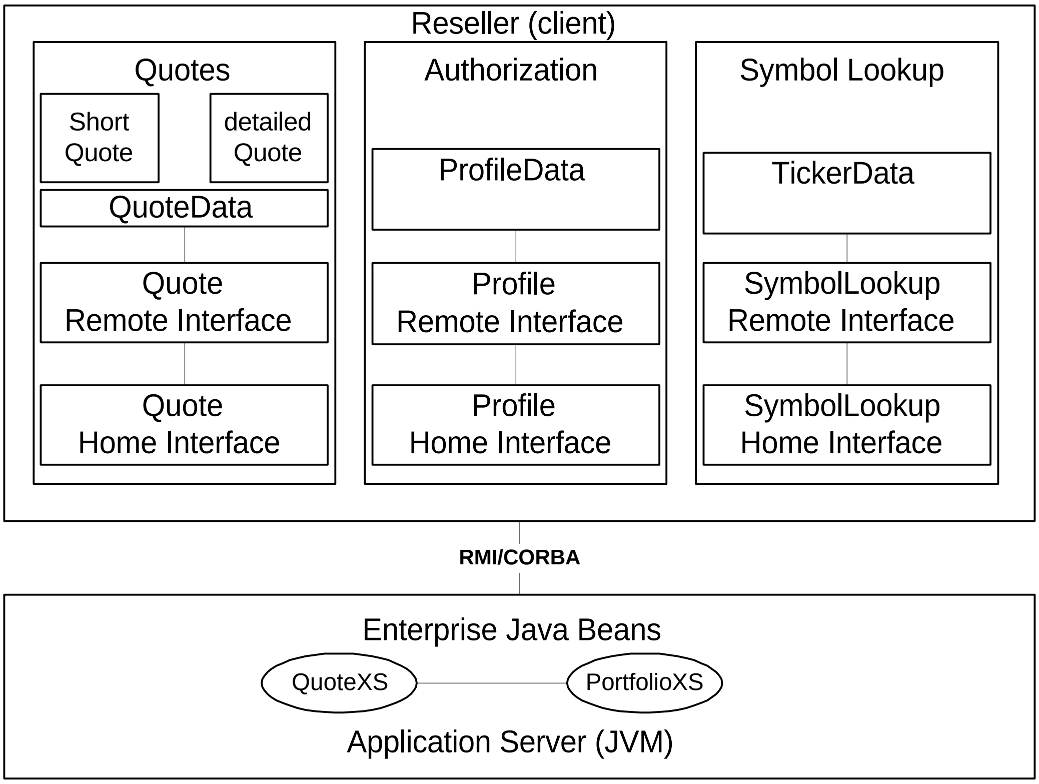
3 The QuoteXS classes

The **QuoteXS** modules have several classes that make our data available to you. They can roughly be divided into three main types of classes.

1. Classes that give you access to quote information:
   QuoteHome; home interface
   Quote; manages the data QuoteData objects
   QuoteData and its subclasses ShortQuote, DetailedQuote; contains the actual delayed/real-time quote information

2. Classes that give you access to user information:
   ProfileHome; home interface
   Profile; manages ProfileData data objects
   ProfileData; contains user information

3. Utility classes that make life easier:
   SymbolLookupHome,
   SymbolLookup,
   TickerData



1.1 How to access the QuoteXS beans?

Typically, our beans our accessed through servlets written in the Java programming language. If you need assistance in writing servlets you may find more information about them on http://java.sun.com/products/servlet/.

Using our JAR files that we provided you with, your servlets will connect to our ***EJBs***. This is done via Java ***RMI*** (Remote Method Invocation). Your servlet must connect to the ***JNDI*** service and do a ***JNDI*** lookup of our bean and then create an instance of the **Quote** bean object. Your servlet can then use this instance to retrieve information about tickers through the object's accessor methods.

(Note: In case you are not using a java-client to connect to the *EJBs* on our server, our application server does support ***CORBA*** too. However, at this point in time this has not been tested)

In short, the following process should followed

1. Create a servlet
2. Log in to the application server
3. Do a JNDI lookup of the bean.
4. Get a reference to the bean using its create() method
5. Get a reference to the QuoteData object
6. Use the object accessor methods to retrieve data from the bean

4 Servlet example

This documentation assumes you're using a servlet to connect to the **MarketXS** beans. This chapter describes how the servlet may connect to the accessor methods as implemented in the **QuoteXS** bean. The Quote Bean 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 **EJBs** via **Java's RMI** to the **MarketXS** application server you will also need the Enterprise Java Bean 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 with 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 be doing in your servlet is log on the 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 our 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 have access to data objects via getQuote() and getQuoteCollection() method. These objects contain the actual financial data.

5 The Quote class (Interface)

1.1 Quote’s Methods

The **Quote** class methods are **getQuote()** for single quotes and **getQuoteCollection()** for retrieving multiple quotes. Both methods return delayed data, unless a user profile object is given. You can specify which objects type you want to be returned by the Quote object by making an instance of the required object and supply it to the Quote object’s getQuote() method or getQuoteCollection() method, along with the tickerId and a possible ProfileData object for real-time quotes.

The Quote object provides access to the data objects mentioned in the QuoteData chapter. The way these objects are retrieved is to supply the Quote object with

- A single tickerId for the getQuote method, or a Collection of tickerIds for the getQuoteCollection method
- The desired data object to return (QuotaData, ShortQuote, DetailedQuote)
- A ProfileData object (optional, for real-time quotes)

Basically, there are 2 types of methods., each with 2 variants.

1. Methods that return delayed quote data (no user profile supplied)
   - getQuote(long tickerId, QuoteData quotedata) (returns a single quote)
   - getQuoteCollection(Collection tickerIdCollection, QuoteData quotedata) (returns multiple quotes)

2. Methods that return real-time data
   - getQuote(long tickerId, QuoteData quotedata, ProfileData profiledata) (will return a single quote)
   - getQuoteCollection(Collection tickerIdCollection, QuoteData quotedata, ProfileData profiledata) (returns multiple quotes)

As you can see, the Quote object is passed a QuotaData object or one of its subclasses., depending on which fields you need for this quote. For QuoteData and its subclasses, see the next chapter.

Example

```
...
 try
 {
  long tickerId = 57575;
  Context ctx = getInitialContext();
  QuoteHome qh = (QuoteHome)ctx.lookup("com.marketxs.quote.Quote");

  Quote quote = qh.create();
  ShortQuote shortquote = quote.getQuote(tickerId, new ShortQuote());

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

…
```

6 The QuoteData class

The Quote object getQuote and getQuoteCollection return a QuoteData object or one of its subclasses. These data objects contain information about the requested tickerId or symbol. The following data objects and their fields are available at present:

Data Object Type	Data Fields	Accessor Method	Description
QuoteData	TickerId	public long getTickerId(l)	Unique key
	CurrencyId	public long getCurrencyId()	Currency
	Symbol	public String getSymbol()	The ticker symbol
	MarketId	public long getMarketId()	Stock Exchange (SE)
	MarketAbbrev	public String getMarketAbbrev()	Abbreviation for SE
	MarketName	public String getMarketName()	Full name for SE
	Name	public String getName()	Name of the fund
ShortQuote (extends QuoteData)	All of QuoteData's fields, plus the following:		
	Last	public Double getLast()	Latest trading price
	CumVol	public Long getCumVol()	Cumulative Volume
	Change	public Long getChange()	Price compared to previous day close
DefaultQuote	All of ShortQuote's fields, plus the following:		
(extends ShortQuote)	Bid		Bidding price
	Ask	public Double getAsk()	Asking price
	High	public Double getHigh()	Highest trading price of the day
	Low	public Double getLow()	Lowest trading price of the day
	Open	public Double getOpen()	Opening price of today
	Yclose	public Double getyClose()	Previous closing trading price
	Updatetime	public String getUpdateTime()	Last trading time
	Delaymins	public Integer getDelayMins()	The delay (lag) in minutes

7 The Profile class (Interface)

The Profile class is used to communicate with the ProfileData class, which holds user information. Profile manages ProfileData data objects. A ProfileData object is a required parameter to supply the Quote objects with 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 functionality:

Functionality	**Method**
See if user is logged in	public boolean isLoggedIn(ProfileData pd, boolean updateTimestamp)
Retrieve a user's profile	public ProfileData getProfile(String loginname)
Add a new user to the reseller database	public add(ProfileData pd)
Update an existing user in the reseller database	public update(ProfileData pd)
Delete an existing user from the reseller database	public delete(ProfileData pd)
Check if a user exists in the reseller database	public boolean exists(ProfileData pd)
Retrieve restrictions to the number of quotes per interval	public Hashtable getQuoteRestrictions(ProfileData pc)
Get the name of the reseller of this user	public String getReseller()

1.1 Updating a user in the database – Example

Example of updating a user in the database (assuming you have done the required *imports*)

```
// 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 (ProfileManager.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");}
```

8 The ProfileData class

The ProfileData class is the class that actually holds the user’s information. It represents a user in the **MarketXS** database. Registered users in our database are allowed to retrieve *real-time* quotes. Anonymous users only have access to **delayed** quotes. When a user of your website wants to access real-time quote, you will have to supply the Quote objects with a ProfileData object, containing the user’s information.

Data Object Type	Data Fields	Accessor Method	Description
ProfileData	profileid	public long getProfileId()	Unique key
	Owner	public boolean isOwner(Secure secure)	Check is if the user owns this profile object
	loginname	public String getLoginname()	Get the user’s login name
		public void setLoginname(String loginname)	Set the user’s login name
	password	public String getPassword()	Get the user’s password
		public void setPassword(String Password)	Set the user’s password
	firstname	public String getFirstname()	Get the user’s first name
		public void getFirstname(String firstname)	Set the user’s first name
	lastnme	public String getLastname()	Get the user’s last name
		public void getLastname(String lastname)	Set the user’s last name
	title	public String getTitle()	Get the user’s title
		public void setTitle(String title)	Set the user’s title
	birthday	public Date getBirthday()	Get the user’s birthday
		public void setBirthday(String title)	Set the user’s birthday
	Addressline1	public String getAddressstreet1()	Get the user’s 1st address line
		public void setAddressstreet1(String address1)	Set the user’s 1st address line
	Addressline2	public String getAddressstreet2()	Get the user’s 2nd address line
		public void setAddressstreet2(String address2)	Set the user’s 2nd address line
	City	public String getCity()	Get the user’s city
		public void setCity(String city)	Set the user’s city
	State	public int getStateId()	Get the user’s state
		public void setStateId(int stateId)	Set the user’s state
	Zipcode	public String getZipcode()	Get the user’s zipcode
		public void setZipcode(String zipcode)	Set the user’s zipcode
	Country	public int getCountryId()	Get the user’s country
		public voidsetCountryId(int countryId)	Set the user’s country
	Email	public String getEmail()	Get the user’s email address
		public void setEmail(String email)	Set the user’s email address
	Phone	public String getPhone()	Get the user’s phone number
		public void getPhone(String phone)	Set the user’s phone number
	SMS	public String getSms()	Get the user’s SMS mobile phone
		public voidgetSms(String sms)	Set the user’s SMS mobile phone
	FirstLogin	public Date getFirstlogin()	Get creation date of this profile

	LastLogin	public Date getLastlogin()	The date this profile was last used (last login date)
		public void setLastlogin(Date lastlogin)	Specifies the date this profile was last accessed.
	Gender	public String getGender()	Get the user’s gender
		public void setGender(String gender)	Set the user’s gender
	QuoteCount	public int getQuotecount()	Get the user’s maximum allowed quotes per interval
		public void setQuotecount(int quotecount)	Sets the user’s maximum allowed quotes per interval

9 The SymbolLookup class

The **SymbolLookup** 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 (**SymbolLookupException**).

1.2 SymbolLookup's Methods

Object	Method	Input	Returns	Example
SymbolLookup	public Collection getTickersBySymbol(String symbol, int type)	Symbol string, representing the stock symbol. May be mixed upper and lower case. A bitmask integer indicating which fields to return (see table)	A collection of tickerIds if the given symbol was found	"ORCL" 5
	public Collection getTickersByName(String company, int type)	A string that represents the name of a company. Must be at least 3 characters. No percentage characters are allowed. The string is used to search our company database for a substring. A bitmask integer indicating which fields to return (see table)	A Collection with tickerid(s) for the given company name. The Collection will hold multiple tickerIds if the company is listen on more than one stock exchange. The Collection is empty when no company was found with that name.	"Oracle" 5

1.3 The integer bitmask for getTickerBySymbol and getTickerByName

Return tickerId for	Binary Value	Decimal Value
Stocks	000001	1
Indices	000010	2
Options	000100	4
Warrants	001000	8
Bonds	010000	16
Miscellaneous	100000	32

For example, if you would like to retrieve the tickerIds for the stocks and options of given symbol, the integer value passed to she method should be 5 (000001 + 000100 = 000101 -> 5).

10 Java Sample Code

The following code implements a servlet that takes one parameter, a tickerId and retrieves the last quote for that tickerid.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

Required for the servlet

```
import javax.ejb.*;
import javax.naming.*;
import java.rmi.*;
```

Required for EJB, RMI and JNDI.

```
import java.util.*;

import com.marketxs.quote.*;
```

Required for the stubs

```
public class basjServlet extends HttpServlet
{
         public void service(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
         {
          PrintWriter out;
          long tickerId;
          Double last;
          String mask = "1";
          Quote q;

          res.setContentType("text/html");
          out = res.getWriter();
          tickerId=req.getParameter("tickerid") == null ? 57257 : new Long(req.getParameter("tickerid")).longValue();

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

Connect to the application server
Look up the QuoteXS EJB in the Naming Directory
Create a Quote object
Create a QuoteData object

```
           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!");}

         } // service

         public Context getInitialContext() throws NamingException
         {
          String user     = "foo";
          String password = "bar";
          String url      = "t3://path.to.appserver.com:7001";
```

Define the username, password and specify the URL where the servlet may find the application server.

```
          Hashtable hash = new Hashtable();
          hash.put(Context.INITIAL_CONTEXT_FACTORY, "weblogic.jndi.WLInitialContextFactory");
          hash.put(Context.PROVIDER_URL, url);
          hash.put(Context.SECURITY_PRINCIPAL, user);
          hash.put(Context.SECURITY_CREDENTIALS, password);
```

Set context properties to log on to the application server

```
          return new InitialContext(hash);
         }

       }
         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;
         }
} // basjServlet
```

11 Glossary

API – Application's **P**rogrammers **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.

HTML - HyperText Markup **L**anguage
HTML is the *lingua franca* 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.

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 - Wireless **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.

JNDI - **J**ava **N**aming and **D**irectory **I**nterface
The Java Naming and Directory Interface^{TM} is a standard extension to the Java^{TM} 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

JDK - **J**ava **D**evelopment **K**it
The Java^{TM} Development Kit (JDK^{TM}) 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

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^{TM} platform and easily integrate and leverage their existing enterprise investments.

JAR - **J**ava **AR**chive
JAR stands for Java ARchive. 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.

Servlet
Servlets are pieces of Java^{TM} 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.

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.

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.

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 Name Service
A network service used to map domain names (e.g. marketxs.com) into IP addresses and vice versa.

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.

RMI – Remote Method Invocation
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.