General Reseller Information

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

1 GENERAL INFORMATION 3

1.1 ROBUST 3 TIER ARCHITECTURE 3
1.1.1 *Tier 1 – Presentation Layer* *4*
1.1.1 *Tier 2 – Business Logic Layer* *4*
1.1.2 *Tier 3 – Data Layer* *4*
1.2 HARDWARE 5
1.3 MODULAR DESIGN 6
1.4 SECURITY ISSUES 7
1.4.1 *Network / Infrastructure security issues (firewalls)* *7*
1.1.2 *Security issues concerning the reseller* *7*
1.1.3 *Security issues concerning the end-user* *7*
1.5 HOW TO ACCESS THE MARKETXS BEANS? 8

2 SERVLET EXAMPLE 9

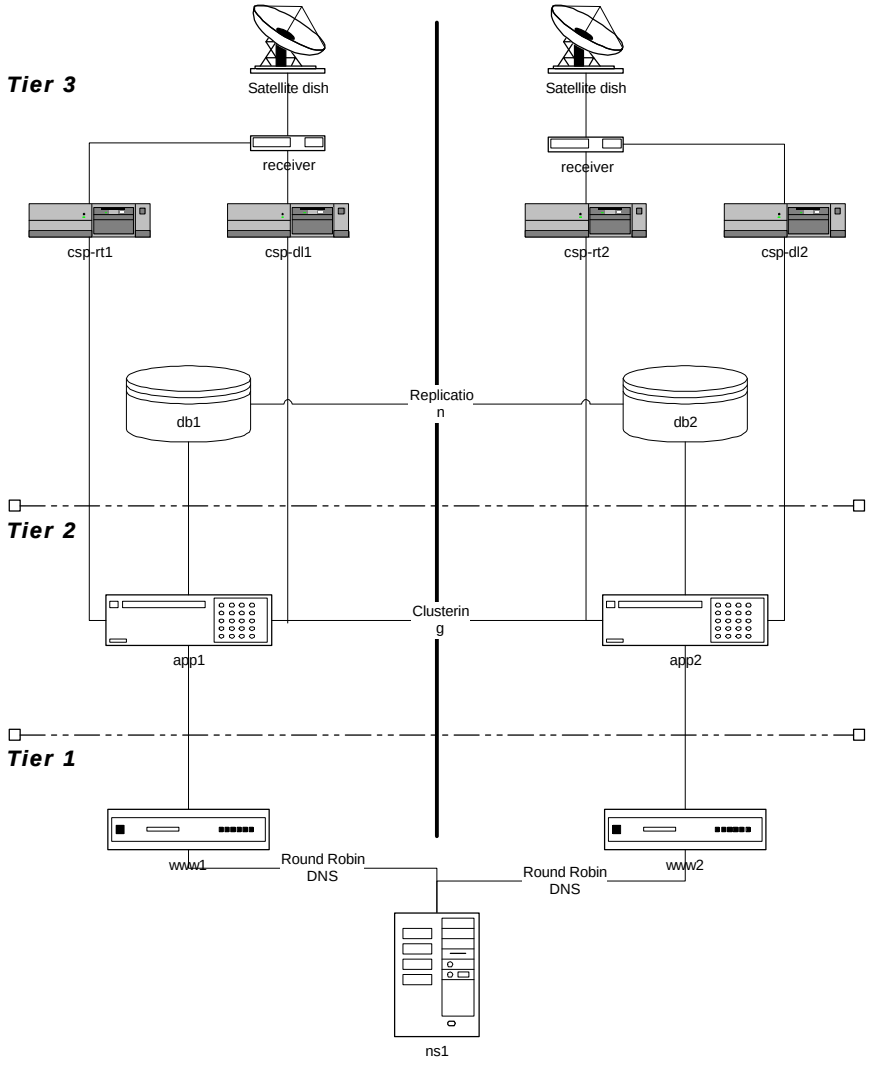
1.1 JAVA SAMPLE CODE 11
1.2 GLOSSARY 12

1 General Information

This document describes the architecture and infrastructure used by **MarketXS** as well as a product description of the classes provided by **MarketXS' QuoteXS** module.

1.1 Robust 3 tier architecture

MarketXS deploys a 3-tier architecture for its applications. Our architecture has proven itself to be robust, scalable and has a high performance. Separating the presentation layer from the business logic and data in 3 tiers enables us to scale wherever necessary. We have chosen industry standard and proven technology components to implement our tiers. The 3-tier design has many advantages. The added modularity makes it easier to modify or replace one tier without affecting the other tiers. In addition, separating the application's functions (business logic) from the database functions (data) makes it easier to implement load balancing. The scheme below of our infrastructure depicts this.



1.1.1 *Tier 1 – Presentation Layer*

Our webservers are utilizing the Linux ***OS***. We have picked Linux as the prefered ***OS*** for our webservers because of its reputation as a webserver. The Linux OS has proven to be very stabile and is able to work under a lot of stress. **MarketXS** prefers using open technology wherever possible. The Linux OS is open, free and yet , does not perform any less than commercial products. It's an excellent platform for creating high performance webservers.

Our Linux webservers run Java ***servlets*** that connect to our middle (business logic) layer. The servlets run in a Java enabled webserver. The servlets connect to the middle tier (business logic layer) which contains our ***Enterprise Java beans*** (EJBs). The EJBs query the database and returns the data to the servlet. On their turn, the servlets fill the templates and send the processed webpage back to the client who requested the webpage.

Using Linux in not a requirement. The only restriction is that your platform should have a ***JVM*** or ***CORBA*** compliant ***ORB*** available to be able to connect to our modules. As our modules are written in the Java programming language, you can access them from any platform that has a JVM implementation.

To spread the load across multiple webservers, a Round Robin ***DNS*** scheme has been implemented. Round Robin DNS is a method of managing server congestion by distributing connection loads across multiple servers (containing identical content). Round Robin works on a rotating basis. One server ***IP address*** is handed out. It then moves to the back of the list of available server IP addresses. Then the next server ***IP address*** is handed out, and it then moves to the end of the list; and so on, depending on the number of servers being used. This works in a looping fashion.

1.1.2 *Tier 2 – Business Logic Layer*

As application server **MarketXS** uses the **WebLogic** application server from BEA (http://www.bea.com) running on Solaris (Sun Microsystems). **WebLogic** implements a Java Virtual Machine version 1.2. Connections to our ***EJBs*** are made through an ***RMI*** (Remote Method Invocation) call. ***RMI*** is Java's native way to access remote objects. The ***API*** documentation gives you access to the functionalities our ***EJBs*** provide.

We have chosen **Enterprise Java Beans** (***EJB***) technology from **Sun Microsystems** (http://www.sun.com) because Sun's ***EJB*** technology is a highly scalable and robust technology. All of our EJB applications are modular by design. For you, as a reseller, this gives you the advantage to expand your business without being worried about performance issues. Our systems scale as you grow.

TODO clustering

1.1.3 *Tier 3 – Data Layer*

As a database we are using the **Oracle 8I** database on Linux. Oracle databases are the industry standard for e-business applications. Using Oracle allows us to cluster and replicate the database for fault-tolerancy and it provides us the necessary scalability for the most demanding e-business applications.

TODO replication!

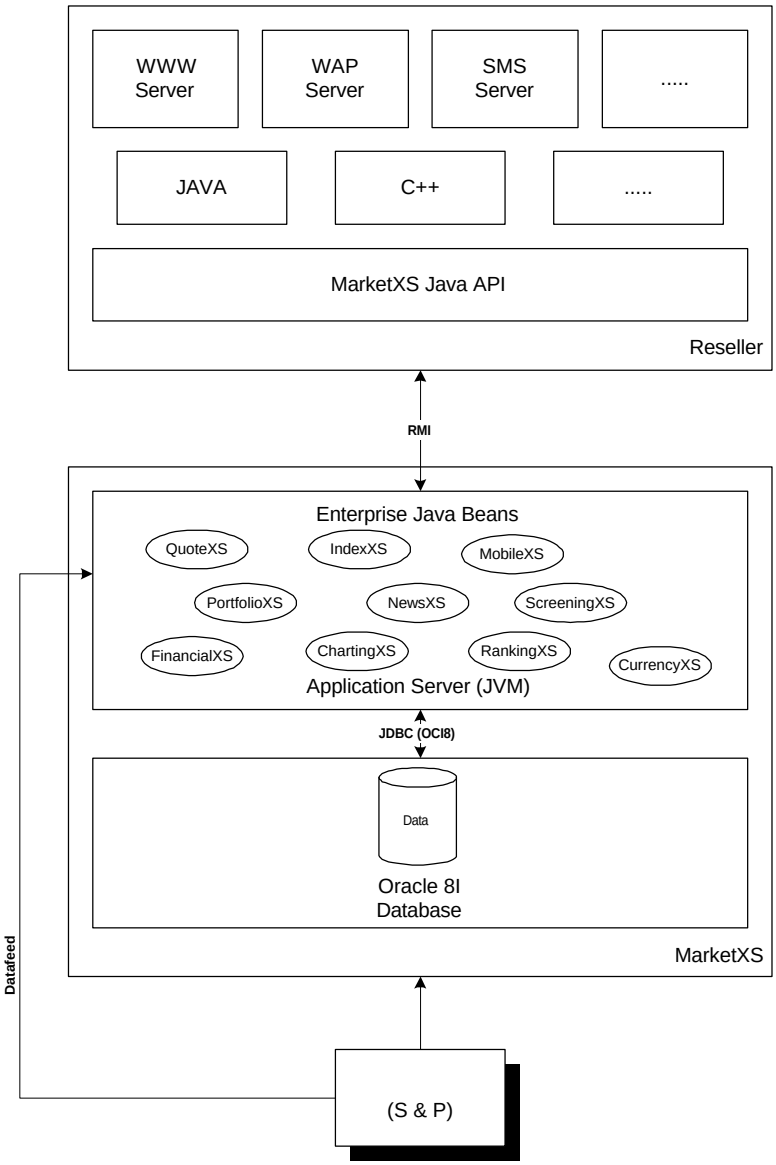
1.2 Hardware

For the hardware we picked **Compaq** and **Sun Microsystems**, both leaders in the market. Our infrastructure has been designed from scratch with fault tolerancy in mind. Therefore, all our systems have built-in redundancy. More redundancy has been achieved through multiple clustered servers. It's designed to serve mission critical applications.

The ***API*** has been generated with Sun's Javadoc tool and is therefore available in ***HTML*** format. Together with the Java classes **MarketXS** provides to the reseller this gives a lot of flexibility to developers as they get access to the data without any restrictions to how the data should be presented in your application. The developers can choose to do whatever they like with the data objects. They can be used for web-based applications (e.g. servlets), desktop applications, ***SMS / WAP*** services. The possibilities are endless. The **MarketXS** beans give you total freedom and control over your applications.

1.3 Modular design

Our modules are designed from the beginning to interact with each other. It is very easy to expand your application's functionality. This enables developers to create powerful and easy to maintain applications. For example, our **QuoteXS** module gives the developer *real-time* and *delayed* quote information from all the exchanges that **MarketXS** has a re-vendor contract with. In combination with **PortfolioXS**, a module used to register users and keep track of their portfolio(s) it becomes possible to create portfolios for users that can display their stock values. The picture below shows in a schematic way how our modules are accessed through ***RMI*** (or ***CORBA***). As you get the raw data objects you can present the date in any way you like (WWW, ***WAP***, ***SMS***).



1.4 Security Issues

Security is of utmost importance to **MarketXS**. A great deal of effort has been done to make sure our network and data is inaccessible to unauthorized users. In addition, a mechanism is in place to make sure your data is only accessible to you and that your user's profiles are safe and inaccessible by other resellers.

1.4.1 *Network / Infrastructure security issues (firewalls)*

To host our servers we the co-location facilities provided by **Level(3) Communications** (http://www.level3.com). **Level (3) Communications** has an advanced fiber optic network optimized for IP data traffic. They have networks in the U.S., Europe and Asia. Their NOC (Network Operations Center) is available 24 hours a day. Their uptime is unparalleled. The NOC monitors the network and intervenes before the situation escalates.

TODO FIREWALLS

In addition, Level (3) has taken precautions to avoid a Distributed Denial of Service attacks (**DDoS**). Level (3)'s router network has already been configured to prevent IP-directed broadcast and IP redirects, thereby making it more resistant to DDoS attacks. In addition, Level (3) currently monitors its production networks for all security anomalies, including DDoS attacks, and takes immediate action when such an attack is identified.

1.1.4 *Security issues concerning the reseller*

To make sure your customer data is safe **MarketXS** has implemented a security mechanism based on random generated security keys. This mechanism enables us to make sure your data is accessible to you only. The profiles of your registered users are kept safe and can not be accessed or deleted by other resellers. This process is transparent to the application developer.

TODO

1.1.5 *Security issues concerning the end-user*

The mechanism mentioned in the previous chapter also enables you to check whether a portfolio actually belongs to the user who is trying to access it. You are not obligated to use this mechanism in your applications. However, it is strongly recommended our clients use this mechanism to ensure that end-users do not delete each other's user profiles.

TODO Also see the **Profile** class' isOwner() method which is used to check whether a portfolio actually belongs to a user. This method should be called before a profile is to be deleted. In that case, you want to make sure the user doing the request is the actual owner of this portfolio.

1.5 How to access the MarketXS 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 bean's object. Your servlet can then use this instance to retrieve information about tickers through the object's accessor methods (getXXX and setXXX methods)

Note: In case you are not using a java-client to connect to the ***EJBs***, our application server does have support for ***CORBA*** too. However, at this point in time this has not been tested is more cumbersome to implement. Connecting to the beans using ***RMI*** is recommended.

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 home interface of bean.
4. Get a reference to the remote interface using its create() method
5. Get a reference to the desired object
6. Use the object methods to retrieve data from the bean

This process is followed in the server example in the next chapter.

1 Servlet example

This documentation assumes you're using a servlet (i.e. java) to connect to the **MarketXS** beans. This chapter describes how the servlet may connect to the accessor methods as implemented in the **QuoteXS** bean but a similar process should be followed for any of the other MarketXS beans. 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 ***JDK*** 1.2 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. The classes in the jar files contain the stubs for the remote objects.

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	"bar"

	server

You can see how these properties are set in the actual java code 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.

1.1 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){ }
         catch (CreateException e){ }

        } // 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;
        } // getQuote
}
```

1.2 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™ 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

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

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.

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™ 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.