Voyager Core JMX Developer's Guide

Version 1.0 for Voyager 8.0

Table of Contents

<This page intentionally left blank>

Introduction

Overview

Java Management Extensions (JMX™) provides a framework for Management Applications to control and monitor Java applications. Applications are managed through interfaces exposed as Management Beans. These Management Beans are similar to standard Java Beans. JMX provides a rich set of features for managing all types of Java applications.

Preface

The purpose of this manual is to provide an introduction to basic JMX features as they pertain to Voyager. This is not meant to be a treatise on JMX. This guide assumes a basic knowledge of Java and distributed computing concepts.

This preface covers the following topics:

- JMX Requirements
- Contacting technical support

JMX Requirements

Before attempting to use the JMX features of Voyager.

- A Java runtime (JRE), J2SE 1.5 or higher, or the Java CDC Personal Profile 1.1 installed on your computer or PDA. For PCs and servers, you can download the latest release of the JDK or JSE from java.sun.com at no charge.
- JMX is bundled with J2SE 1.5.

Contacting Technical Support

Recursion Software welcomes your problem reports, and appreciates all comments and suggestions for improving VOYAGER. Please send all feedback to the Recursion Software Technical Support department.

Technical support for VOYAGER is available via the web, email, and phone. You can contact Technical Support by sending email to psupport@recursionsw.com or by calling (972) 731-8800.

JMX Overview

Types of Management Beans (MBeans)

There are four types of MBeans: Standard, Dynamic, Model, and Open. All Voyager MBeans are currently Standard MBeans. Further information concerning the types of MBeans can be obtained from the JMX specification. A Standard MBean explicitly defines its management interface by following naming conventions called design patterns in the JMX specification. The naming conventions are similar to the JavaBeans component model. However, the conventions in JMX take into account the inheritance scheme of the MBean. Standard MBeans are much simpler for management applications to access when compared with other types of MBeans. MBeans provide access to application attributes (getter/setters) and methods to perform control operations.

The MBeanServer

The MBeanServer is a key concept in JMX. The server object acts as a container of MBeans and provides a common interface that all management applications must use to access the MBeans. An *ObjectName* uniquely identifies each MBean within the server object. The ObjectName is comprised of a domain name and application defined attributes. The domain name usually follows the Java package name prefix conventions. All Voyager MBeans have a domain name of *com.recursionsw.ve*.

Notifications

MBeans provide access to attributes and facilitate control operations. In addition, they can also emit asynchronous notifications. This is a publish/subscribe style of messaging. Interested receivers can register to be notified of particular events by specifying a filter, or they can receive all events emitted by an MBean instance. Most of the Voyager MBeans support notifications for relevant events.

Voyager MBeans

The following table lists the available MBeans for Voyager, and indicates their ObjectName key properties. All Voyager MBeans are registered in the com.recursionsw.ve domain. A detailed discussion of each MBean follows.

MBean (type=<column value>, vm=<Voyager instance id>)	*additional ObjectName key properties*
VoyagerAdmin	
AgentPeer	peerGroupName=<name of containing PeerGroup>, peerName=<assigned name>
AgentSpace	name=<assigned name>
PeerGroupManager	agentSpaceName=<name of containing AgentSpace>, peerGroupName=<assigned name>
AuditServiceConsoleLogService	
AuditServiceFileLoggerService	
AuditService	
DirectoryNamingService	
ORBServer	url=<quoted url of associated transport server>
Routing	
TcpSubspace	name=<assigned name>, guid
TcpTransport	
UdpTransport	
DGC YellowPages	

Figure 1: Voyager MBean ObjectName key properties

VoyagerAdmin MBean

The VoyagerAdmin MBean provides the management behaviors for a Voyager instance. It provides the capability to start and stop Voyager, configure Voyager, and register for log events.

Naming

The name of the VoyagerAdmin MBean contains the “type” and “vm” attributes.

Lifecycle

The VoyagerAdmin MBean is registered during Management.startup() and deregistered during Management.shutdown().

MBean

```
void assignPropertyValue(String property, String value);
boolean checkLogTopicEnabled(String topic);
void disableLogTopic(String topic);
void disableLogTopics(String[] topics);
void enableLogTopic(String topic);
void enableLogTopics(String[] topics);
String getPropertyFileName();
String getStartupUrl();
String getUrl();
String getVersion();
boolean isStarted();
String [] retrieveAllProperties();
String retrievePropertyValue(String property);
void setPropertyFileName(String newPropertyFileName);
void setUrl(String newUrl);
void start();
void stop();
```

Emitted Events

ve.log.topic.<topic name> - for each log event for the given topic (logging for that topic must be enabled in order to receive notifications for that topic).

AuditService MBean

The AuditService MBean provides configuration and control of the Voyager audit subsystem. The management operations support adding remote Voyager audit subsystems as parents or peers and breaking an established relationship.

See also the MBeans for managing the console and file loggers.

Naming

The name of the AuditService MBean contains the “type” and “vm” attributes.

Lifecycle

The AuditService MBean is registered during Voyager.startup() and deregistered during Voyager.shutdown().

MBean

```
void connectParent(String parentURL);
void connectPeer(String peerURL);
void disconnectNeighbor(String neighborURL);
String getConnectionPolicy();
int getNeighborCount();
String[] getParentURLs();
String[] getPeerURLs();
void setConnectionPolicy(final String className)
      throws ClassNotFoundException;
void stop();
void start();
```

Emitted Events

`ve.audit.service.enabled` – the AuditService is enabled.
`ve.audit.service.disabled` – the AuditService is disabled.
`jmx.attribute.change` – when the connection policy is changed.

AuditServiceConsoleLogService MBean

The AuditServiceConsoleLogService MBean provides management of the service that writes audit records to the Voyager console. Its primary capability is to control the filtering of the audit records output to the console.

Naming

The name of the AuditServiceConsoleLogService MBean contains the "type" and "vm" attributes.

Lifecycle

The AuditServiceConsoleLogService MBean is registered during Voyager.startup() and deregistered during Voyager.shutdown().

MBean

```
void addFilter(int outcome, boolean filterOn);
boolean getPrintDenialRecords();
boolean getPrintFailureRecords();
boolean getPrintSuccessRecords();
int getRecordCount();
void resetFilters();
void setPrintDenialRecords(boolean enable);
void setPrintFailureRecords(boolean enable);
void setPrintSuccessRecords(boolean enable);
void start();
void stop();
void resetFilters()
```

Emitted Events

`ve.audit.service.enabled` – the AuditServiceConsoleLogService is enabled.
`ve.audit.service.disabled` – the AuditServiceConsoleLogService is disabled.
`jmx.attribute.change` – when an attribute of AuditServiceConsoleLogService has changed.
`ve.audit.filterchange` – The console logger's filter configuration has changed.

AuditServiceFileLoggerService MBean

The audit file logging service, which is optional and not a default service, writes records received by the Audit system to a file. Unlike the console log service, the audit file logging service implements no filtering. The MBean that manages this service can configure the directory for the log files, as well as the prefix and suffix to use for the files. The MBean also indicates the current and previous audit files and the current record count.

Naming

The name of an AuditServiceFileLoggerService MBean contains the "type" and "vm" attributes.

Lifecycle

The MBean is created when the audit file logger is configured. The audit file logger service is optional and not, by default, created and initialized. The service is destroyed and its MBean deregistered when Voyager shuts down.

MBean

```
String getCurrentFileName();
String getDefaultFileNamePrefix();
String getDefaultFileNameSuffix();
String getDirectory();
String getPreviousFileName();
long getRecordCount();
void setDefaultFileNamePrefix(String aFileNamePrefix);
void setDefaultFileNameSuffix(String aFileNameSuffix);
void setDirectory(String directoryName);
void stop();
void startNewLogFile();
void start();
```

Emitted Events

`ve.audit.service.enabled` – the AuditServiceFileLoggerService is enabled.

`ve.audit.service.disabled` – the AuditServiceFileLoggerService is disabled.
`jmx.attribute.change` – when an attribute of AuditServiceFileLoggerService has changed or the file logger has closed the old file, if any, and changed to a new file.

Distributed Garbage Collection MBean

The Distributed Garbage Collection (DGC) MBean controls Voyager's distributed garbage collection service.

Naming

The DGC object name contains the "type" and "vm" attributes.

Lifecycle

The DGC MBean is registered when Voyager starts, and deregistered when Voyager shuts down.

MBean

```
long getLastCycleTime();
long getCycleTime();
void setCycleTime(long milliseconds);
long getDiscardDelay();
void setDiscardDelay(long milliseconds);
boolean getForceGC();
void setForceGC(boolean forceGC);
boolean getForceFinalization();
void setForceFinalization(boolean forceFinalization);
int getImportCount();
int getPendingCount();
String[] getImportXurls();
String[] getExportXurls();
String[] getExportsFor(String xurl);
String getImportActionsFor(String xurl);
```

Emitted Events

`ve.messageprotocol.vrmp.dgc.discarding` – An object is being garbage collected.
`ve.messageprotocol.vrmp.dgc.exporting` – An object has been registered with DGC (DGC enabled).

TcpTransport MBean

The TcpTransport MBean serves to monitor, configure, and control the TcpTransport service and its related connections and servers.

Naming

The TcpTransport MBean object name contains the “type” and “vm” attributes.

Lifecycle

The TcpTransport MBean is registered during Voyager.startup() and deregistered during Voyager.shutdown().

MBean

```
void closeConnections(String xurl);
int getActiveCount();
String[] getConnectionURLs();
int getIdleCount();
String getProtocol();
String[] getServerURLs();
int retrieveActiveCount(String xurl);
int retrieveIdleCount(String xurl);
void stop();
void start();
void startServer(String xurl) throws IOException;
void stopServer(String xurl);
void addMultiHomeHost(String host);
```

Emitted Events

`ve.transport.server.stopped` – A TCP transport server has been stopped.
`ve.transport.server.started` – A TCP transport server has been started.
`ve.transport.server.connection.failed` – TCP server side connection failed.
`ve.transport.connection.opened` – A TCP transport connection has been opened.
`ve.transport.connection.closed` – A TCP transport connection has been closed.
`ve.transport.connection.released` – A TCP transport connection was released.
`ve.transport.connection.failed` – A TCP transport connection failed.
`ve.transport.connection.acquired` – A TCP transport connection was acquired.

Routing MBean

The Routing MBean serves to monitor, configure, and control the Routing service and its related connections and servers.

Naming

The Routing MBean object name contains the “type” and “vm” attributes.

Lifecycle

The Routing MBean is registered during Voyager.startup() if routing is enabled and deregistered during Voyager.shutdown().

MBean

```
void closeConnections(String xurl);
int getActiveCount();
String[] getConnectionURLs();
int getIdleCount();
String getProtocol();
String[] getServerURLs();
int retrieveActiveCount(String xurl);
int retrieveIdleCount(String xurl);
void stop();
void start();
void startServer(String xurl) throws IOException;
void stopServer(String xurl);
void addMultiHomeHost(String host);
String getRouterAddress();
```

Emitted Events

`ve.transport.server.stopped` – A Routing transport server has been stopped.
`ve.transport.server.started` – A Routing transport server has been started.
`ve.transport.server.connection.failed` – Routing server side connection failed.
`ve.transport.connection.opened` – A Routing transport connection has been opened.
`ve.transport.connection.closed` – A Routing transport connection has been closed.
`ve.transport.connection.released` – A Routing transport connection was released.
`ve.transport.connection.failed` – A Routing transport connection failed.
`ve.transport.connection.acquired` – A Routing transport connection was acquired.

UdpTransport MBean

The UdpTransport MBean serves to monitor, configure, and control the UdpTransport service and its related connections and servers.

Naming

The UdpTransport MBean object name contains the “type” and “vm” attributes.

Lifecycle

The UdpTransport MBean is registered during Voyager.startup() and deregistered during Voyager.shutdown().

MBean

```
void closeConnections(String xurl);
int getActiveCount();
String[] getConnectionURLs();
int getIdleCount();
String getProtocol();
String[] getServerURLs();
int retrieveActiveCount(String xurl);
int retrieveIdleCount(String xurl);
void stop();
void start();
void startServer(String xurl) throws IOException;
void stopServer(String xurl);
int getMaximumMessageSize();
int getServerMessageBufferSize();
String getSocketPolicyManagerClass();
void setMaximumMessageSize(int size);
void setServerMessageBufferSize(int bufferSize);
void setSocketPolicyManagerClass(String className)
      throws ClassNotFoundException;
```

Emitted Events

`ve.transport.server.stopped` – A UDP transport server has been stopped.
`ve.transport.server.started` – A UDP transport server has been started.
`ve.transport.server.connection.failed` – UDP server side connection failed.
`ve.transport.connection.opened` – A UDP transport connection has been opened.
`ve.transport.connection.closed` – A UDP transport connection has been closed.
`ve.transport.connection.released` – A UDP transport connection was released.
`ve.transport.connection.failed` – A UDP transport connection failed.
`ve.transport.connection.acquired` – A UDP transport connection was acquired.

DirectoryNamingService MBean

The DirectoryNamingService MBean provides monitoring and management for Voyager's primary naming service.

Naming

The DirectoryNamingService object name contains the "type" and "vm" attributes.

Lifecycle

The DirectoryNamingService MBean is registered when Voyager starts, and deregistered when Voyager shuts down.

MBean

```
String[] getKeys(String lookupkey)
void unbind(String lookupkey)
```

Emitted Events

`ve.directory.bind` – A directory entry was bound.
`ve.directory.rebind` – A directory entry was rebound.
`ve.directory.unbind` – A directory entry was unbound.

YellowPages MBean

The Yellow Pages Directory service provides a mapping between a service description, consisting of one or more name-value service attributes, and a service. The YellowPages MBean provides configuration and control of a YellowPages instance (a member of the federated Yellow Pages directory service).

Naming

The YellowPages object name contains the "type" and "vm" attributes.

Lifecycle

The YellowPages MBean is registered when Voyager starts, and deregistered when Voyager shuts down.

MBean

```
void disconnect();
int getNeighborCount();
int getServiceCount();
long getDefaultDiscoveryRequestTTL();
```

```
long getDiscoveryRequestMarkerExpirationDelay();
void setDefaultDiscoveryRequestTTL(long delay);
void setDiscoveryRequestMarkerExpirationDelay(long delay);
void terminateAllDiscoveryRequests();
```

Emitted Events

`ve.yp.serviceregistration` – A service description was added or removed.
`ve.yp.connection`– A connection to a remote YellowPages directory was opened or closed.

ORBServer MBean

The object request broker (ORB) is created by and contained within a transport server. The ORBServer MBean provides monitoring and control of an ORB instance.

Naming

The ORBServer object name contains the "type", "vm", and "url" attributes (the URL being the URL of the transport server).

Lifecycle

The managed object is created at the same time as the containing transport server, and destroyed when the containing transport server is destroyed. The MBean for the ORB follows the ORB's lifecycle.

MBean

```
boolean checkRegistered(int id);
int getRegisteredCount();
String getServerUrl();
TabularData retrieveSummaryInfo();
void unregister(int id);
```

Emitted Events

None.

TcpSubspace MBean

A Voyager Space is a concept realized by one or more Subspaces. The TcpSubspace MBean manages a TcpSubspace (a Subspace of a Space whose connections are based on TCP).

Naming

The TcpSubspace object name contains the “type”, “vm”, “name”, and “guid” attributes. If the space is unnamed, the “name” attribute value will be the string “Anonymous Space”.

Lifecycle

TcpSubspaces are created and destroyed by the application and, for internal use, by Voyager. The MBean for the TcpSubspace follows the TcpSubspace’s lifecycle.

MBean

```
int getListenerCount();
int getMemberCount();
int getNeighborCount();
int getMarkerCount();
long getMarkerLifetime();
int getQueueSize();
long getTimeOfLastPurge();
void purgePropagationQueue();
```

Emitted Events

`ve.space.add` – An object was added to the subspace.
`ve.space.connected` – A neighbor was connected to the subspace.
`ve.space.connecting` – A neighbor is being connected to the subspace.
`ve.space.disconnected` – A neighbor was disconnected from the subspace.
`ve.space.disconnecting` – A neighbor is being disconnected from the subspace.
`ve.space.purging` – The subspace is purging dead/disconnected neighbors, children, and contents.
`ve.space.removing` – An object was removed from the subspace.

PeerGroupManager MBean

The managed object, a PeerGroupManager instance, is responsible for managing the AgentPeers in a PeerGroup. The PeerGroupManager MBean provides monitoring of, and the ability to terminate, the PeerGroupManager.

Naming

The PeerGroupManager object name contains the “type”, “vm”, “agentSpaceName” and “peerGroupName” attributes.

Lifecycle

A PeerGroupManager is created and destroyed by the application and its MBean follows its lifecycle.

MBean

```
String getAgentSpaceName();
String [] getAgentPeerNames();
String getPeerGroupName();
int getAgentPeerCount();
long getLastHeartbeatTime();
long getCyclesBeforeDisconnection();
long getCycleTime();
long getLastCycleTime();
void stop();
```

Emitted Events

`ve.agentspace.events.agentpeerdisconnectedevent` – An Agent Peer has been disconnected from the peergroup due to connection failure.
`ve.agentspace.events.agentpeerjoiningevent` – An Agent Peer is joining the peergroup.
`ve.agentspace.events.agentpeerleavingevent` – An AgentPeer is leaving the peergroup.
`ve.agentspace.events.peergroupjoiningevent` – The PeerGroupManager is joining an AgentSpace.
`ve.agentspace.events.peergroupleavingevent` – The PeerGroupManager is leaving an AgentSpace.

AgentPeer MBean

The managed object, an AgentPeer, hosts and executes agents, receiving them from the PeerGroupManager it is registered with. An "agent" is an instance of the Agent class with an application-specified realization of IAgentAction. The MBean for an AgentPeer provides monitoring of the AgentPeer and limited control of the AgentPeer and associated agents.

Naming

The AgentPeer MBean object name contains the "type", "vm", "peerGroupName" and "peerName" attributes.

Lifecycle

An AgentPeer is created and destroyed by the application. The MBean follows the lifecycle of the AgentPeer it manages.

MBean

```
int getAgentCount();
String getGroupName();
String getPeerName();
String[] getAgentGuids();
String getAgentInfo(String guid);
void interruptAgentExecution(String guid);
void stop();
```

Emitted Events

`ve.agentspace.events.agentarrivalevent` – An agent has arrived at the peer.
`ve.agentspace.events.agentexecutionendevent` – An agent has finished execution.
`ve.agentspace.events.agentexecutionerrorevent` – An internal error occurred during deployment/execution of an agent.
`ve.agentspace.events.agentexecutionfailureevent` – An executing agent failed due to an agent exception or timeout.
`ve.agentspace.events.agentexecutionstartevent` – An agent has started execution.
`ve.agentspace.events.agentexecutiontimeoutevent` – An executing agent has timed out.
`ve.agentspace.events.agentscheduledevent` – An agent has been scheduled for execution.
`ve.agentspace.events.agentpeerheartbeatevent` – The agent peer is notifying its PeerGroupManager it is alive.

AgentSpace MBean

The managed object, an AgentSpace, provides the primary interface for interacting with an AgentSpace.

Naming

The AgentSpace MBean object name contains the “type”, “vm”, and “name” attributes.

Lifecycle

An AgentSpace is created and destroyed by the application. The MBean for the AgentSpace follows the AgentSpace’s lifecycle.

MBean

```
String getAgentSpaceName();
String[] getPeerGroupNames();
void stop();
```

Emitted Events

`ve.agentspace.events.createpeergroupmanagerevent` – A PeerGroupManager has been created.
`ve.agentspace.events.createagentpeerevent` – An AgentPeer has been created.
`ve.agentspace.events.createagentevent` – An agent has been created.

Management of User Agents

User agents are required to implement the IManagedAgent interface (which serves to provide the management bean for the agent). The associated MBean is required to implement the IAgentMx[1] interface.

A managed application-specific agent can be any of the following.

- A Java object that has been exported
- An AgentSpace action
- A class with an agent facet

Naming

The application-specific agent's MBean domain name, returned by IAgentMx's getDomainName(), cannot begin with "com.recursionsw". Voyager imposes no other restrictions on the MBean name.

IManagedAgent interface

An AgentFacet agent that wants to be managed automatically should implement this interface. It defines a single method:

```
IAgentMx getManagementBean();
```

This method returns the management agent for the agent.

IAgentMx interface

```
String getDomainName();
com.recursionsw.lib.container.Pair[] getKeyProperties();
```

Emitted Events

Voyager will emit no events related to an application-specific Agent or Agent MBean. The application-specific Agent is free to emit application-specific events.

[1] The MBean's management interface should not extend IAgentMx, the bean should implement IAgentMx as well as its management interface.

Example

To illustrate how an object used as an agent might be managed; we will add the necessary interfaces and classes to the current Voyager Agents1 example. The `Agents1` example uses the stock market `Trader` class as the agent. To the `Trader` class needs to implement the `IManagedAgent` interface. We must also create the related MBean interface and implementation, which we will name ManageTrader and `ManageTraderMBean`, respectively.

The class diagram in Figure 2 shows the relationships among these classes.

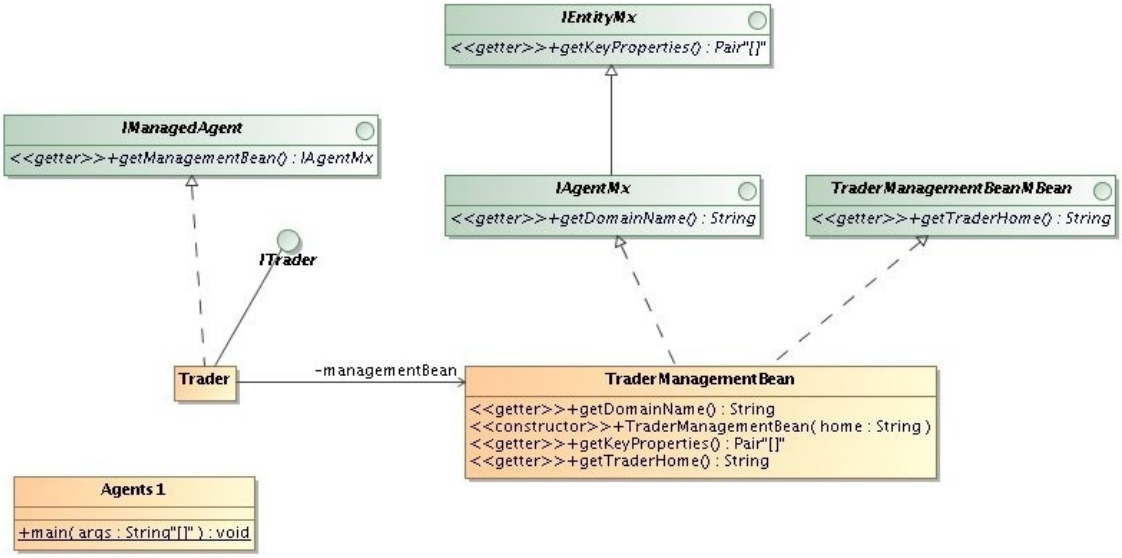

Figure 2: Class diagram for managed Agents1 example

The sequence diagram shown in Figure 3 illustrates both setting up the agent for management and, as the agent cleans up, disabling management of the agent.

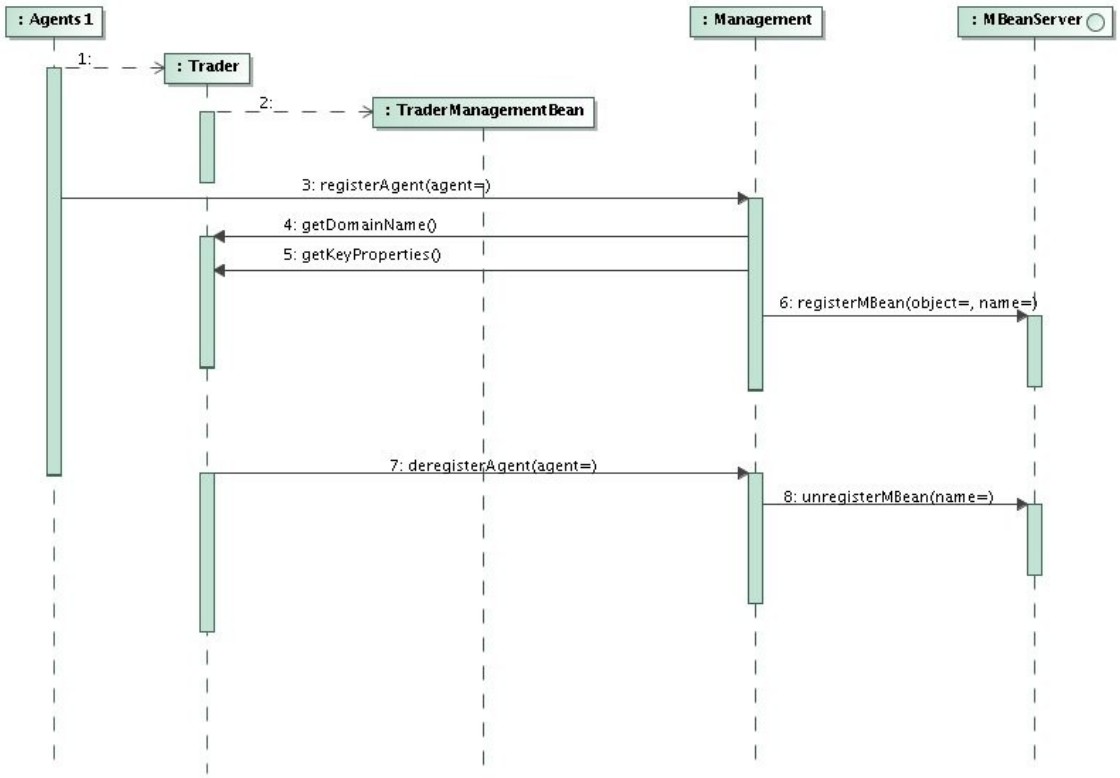

Figure 3: Start/Stop managing the Agents1 example.

Initializing JMX Support

Voyager's JMX support must be initialized before it can be used. To do this, call one of the com.recursionsw.ve.jmx.Management.startup() methods prior to calling Voyager.startup()[2]. An application using JXM must set Voyager to default to Java's serialization. Other serialization options may or may not support serialization of JMX objects.

Your application has the choice of supplying the MBeanServer to be used or of allowing Voyager to locate or create the MBeanServer to be used. For example:

```
import javax.management.MBeanServer;

import com.recursionsw.ve.StartupException;
import com.recursionsw.ve.Voyager;
import com.recursionsw.ve.jmx.Management;

…
```

[2] Please note that the ve-jmx.jar archive must be in the classpath

```
try
{
  MBeanServer mbs = Management.startup();
  VrmpSerialization.selectJavaDefaultSerialization();
  Voyager.startup();
  ...
}
catch (StartupException e)
{
  e.printStackTrace();
}
```

This code performs the following:

- Locates or creates an MBeanServer (uses the platform MBeanServer if available)
- Registers a VoyagerAdminMBean instance in the MBeanServer
- Sets the default object serialization implementation to Java
- Initializes Voyager

The VoyagerAdminMBean can now be accessed via the MBeanServer object. Invoking MBean methods can be cumbersome due to the reflective invoke() method of the MBeanServer. However, if Standard MBeans are being used (which is the case with Voyager) then a convenient proxy can be created to ease MBean invocation. For example (modified from JMXExample2):

```
Management.startup();
VrmpSerialization.selectJavaDefaultSerialization();
VoyagerContext vc = Voyager.startup();
ServerContext serverContext = vc.acquireServerContext("server8000");
serverContext.startServer("//localhost:8000");
ClientContext clientContext = vc.acquireClientContext("server80001");
clientContext.openEndpoint("//localhost:8001");
try {
  IVoyagerAdmin admin = Management.getVoyagerAdmin(clientContext);
  MBeanServerConnection remoteServer = admin.getMBeanServer();
  if (remoteServer != null) {
    Set set = remoteServer.queryNames(
                new ObjectName(Management.DEFAULT_DOMAIN +
                               ":type=TcpTransport,*"), null);
    Iterator iter = set.iterator();
    if (iter.hasNext()) {
      ObjectName name = (ObjectName) iter.next();
      TcpTransportMxMBean transportMBean = (TcpTransportMxMBean)
          JMXUtil.newProxy(remoteServer, name,
                           TcpTransportMxMBean.class);
      String [] urls = transportMBean.getServerURLs();
      …
    }
  }
} catch (Exception e) {
  e.printStackTrace();
```

```
} finally {
  vc.shutdown();
  Management.shutdown();
}
```

This example initializes the JMX support, starts Voyager, accesses the MBeanServer of another managed Voyager instance, searches for the remote Voyager's TcpTransport MBean, and invokes the getServerURLs() method on the located MBean.

Using JConsole

JConsole is a JMX management console that ships with Java 5 and higher. It allows a user to view and modify MBean attributes, perform invocations on MBean methods, and view event notifications. One running instance of JConsole can manage multiple nodes. In order to remotely manage a node, that node must have a JMXConnectorServer running. A JMXConnector server can be started in one of three ways:

1. Programmatically:

```
/* Example of starting an RMI connector using Voyager as the RMI
 * registry and JNDI provider. */

import java.util.Properties;
import javax.management.MBeanServer;
import javax.management.remote.JMXConnectorServer;
import javax.management.remote.JMXConnectorServerFactory;
import javax.management.remote.JMXServiceURL;
import com.recursionsw.ve.Voyager;
import com.recursionsw.ve.jmx.Management;

public class JMXConnectorServerTest {

    public static void main(String[] args) throws Exception {
    MBeanServer server = Management.startup();
    VoyagerContext vc = Voyager.startup();
    ServerContext serverContext = vc.acquireServerContext("server8000");
    serverContext.startServer("//myhost:8000");
    Properties environment = new Properties();
    environment.put("java.naming.factory.initial",
        "com.recursionsw.ve.jndi.spi.VoyagerContextFactory");
    environment.put("java.naming.provider.url", "//myhost:8000/");

    // The address of the connector server
    JMXServiceURL address = new JMXServiceURL(
        "service:jmx:rmi:///jndi/rmi://myhost:8000/server");

    // Create the JMXCconnectorServer
    JMXConnectorServer cntorServer =
        JMXConnectorServerFactory.newJMXConnectorServer(
                address, environment, server);
```

```
    // Start the JMXConnectorServer
    cntorServer.start();
  }
}
```

2. Via command line options (when the platform MBeanServer is used)

```
To enable remotely (without SSL):

java -Dcom.sun.management.jmxremote
-Dcom.sun.management.jmxremote.port=<port>
-Dcom.sun.management.jmxremote.ssl=false MyApp

To connect locally only:

java -Dcom.sun.management.jmxremote
```

3. Via dynamic attach
   a. The Java 6 version of JConsole must be used.
   b. The node must be running Java 6.
   c. The node to be managed must be local.
   d. The platform MBeanServer must be used.

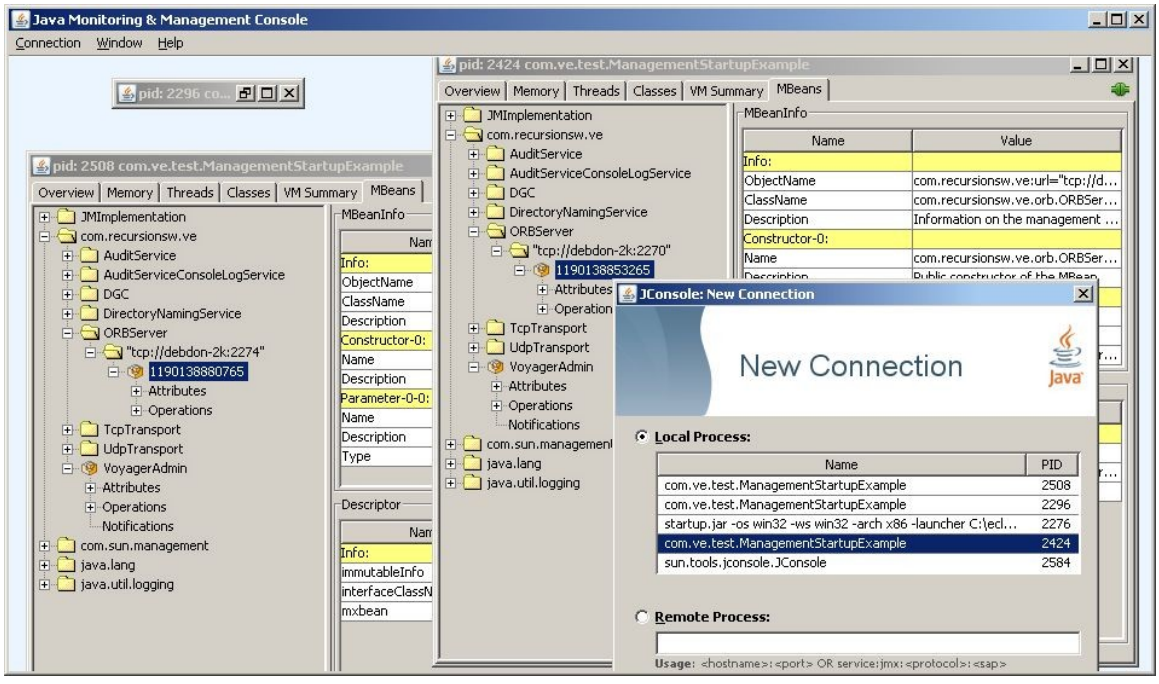

Figure 4: JConsole (Java 6) managing multiple VMs

For more information on using JConsole, please refer to
http://java.sun.com/javase/6/docs/technotes/guides/management/jconsole.html.

JMX Examples

The JMX examples demonstrate initialization of Voyager JMX management, access to local and remote Voyager MBeans, and implementation of managed agents. These examples require Java 5 or higher, or a Java 1.4 runtime with JMX 1.2 installed.

JMXExample1

The `JMXExample1` program demonstrates JMX management initialization and access to local Voyager MBeans, specifically the TCP or Bidi transport in this case.

JMXExample2

The `JMXExample2a` and `Basics2b` programs demonstrate access to remote Voyager MBeans using the IVoyagerAdmin interface, specifically the TCP or Bidi transport in this case.

ManagedAgents

The `ManagedAgents1` program illustrates creating and using an agent with JMX management support. It is essentially a clone of the Agents1 example.

Source code location

The examples can be found under the %VOYAGER_HOME%\examples\java\se-cdc directory.

`Java`

java\examples\jmx\JMXExample1.java

java\examples\jmx\JMXExample2a.java
java\examples\jmx\JMXExample2b.java

java\examples\managedagents\ITrader.java
java\examples\managedagents\Trader.java

java\examples\managedagents\TraderManagementBean.java
java\examples\managedagents\TraderManagementBeanMBean.java

java\examples\managedagents\ManagedAgents1.java