

SDN Home > Products & Technologies > Java Technology > Reference > Technical Articles and Tips > Developer Technical Articles & Tips >

Article

Getting Started with Java Management Extensions (JMX): Developing Management and Monitoring Solutions

By Qusay H. Mahmoud, January 6, 2004

Contents

- Overview of JMX Technology
- JMX Tiered Architecture
- JMX Technology Implementations
- JMX Technology and J2SE 5.0
- The JMX Technology Programming Model
- JMX Technology for Remote Management
- JMX Technology Related Specifications

The Java Management Extensions (JMX) API is a standard —developed through the [Java Community Process](#) (JCP) as [JSR 3](#)—for managing and monitoring applications and services. It defines a management architecture, design patterns, APIs, and services for building web-based, distributed, dynamic, and modular solutions to manage Java-enabled resources. The JMX APIs make it possible to add manageability to Java-enabled equipment, from web phones to set-top boxes to network devices and servers. Using JMX technology to manage applications and services increases their value to vendors and clients by making applications easier to install, configure, and maintain.

This article provides a fast track technical tutorial to JMX technology. It discusses JMX's tiered architecture, the JMX programming model, and code demonstrating how to use JMX technology to develop management applications. In addition, the article shows how to use a JMX technology-compliant management tool in the Java 2 Platform, Standard Edition 5.0 (J2SE 5.0), which has implemented version 1.2 of the JMX specification.

Overview of JMX Technology

The JMX technology is native to the Java programming language. As a result, it offers natural, efficient, and lightweight management extensions to Java-based functions. It consists of a set of specifications and development tools for managing Java environments and developing state-of-the-art management solutions for applications and services. It provides Java developers with the means to instrument Java code, create smart Java agents, implement distributed management middleware and managers, and easily integrate these solutions into existing management and monitoring systems. The dynamics of the JMX technology architecture enables you to use it to monitor and manage resources as they are implemented and installed. It can also be used to monitor and manage the [Java Virtual Machine \(JVM machine\)](#).

Typical uses of the JMX technology include:

- Consulting and changing application configuration



Feedback

- Collecting statistics about application behavior and making the statistics available
- Notification of state changes and erroneous conditions

Benefits of JMX Technology

The JMX technology enables Java developers to encapsulate resources as Java objects and expose them as management resources in a distributed environment. The JMX specification lists the following benefits to using it to build a management infrastructure:

- *Manages Java applications and services without heavy investment:* JMX architecture relies on a core managed object server that acts as a management agent and can run on most Java-enabled devices. Java applications can be managed with little impact on their design.
- *Provides a scalable management architecture:* A JMX agent service is independent and can be plugged into the management agent. The component-based approach enables JMX solutions to scale from small footprint devices to large telecommunications switches.
- *Can leverage future management concepts:* It can implement flexible and dynamic management solutions. It can leverage emerging technologies; for example JMX solutions can use lookup and discovery services such as Jini network technology, UPnP, and Service Location Protocol (SLP).
- *Focuses on management:* While JMX technology provides a number of services designed to fit into a distributed environment, its APIs are focused on providing functionality for managing networks, systems, applications, and services.

JMX Tiered Architecture

JMX technology provides a tiered architecture where managed resources and management applications can be integrated in the plug-and-play approach as shown in Figure 1. A given resource is instrumented by one or more Java objects known as Managed Beans (or MBeans), which are registered in a core managed object server known as the MBean server. This server acts as a management agent and can run on most Java-enabled devices.

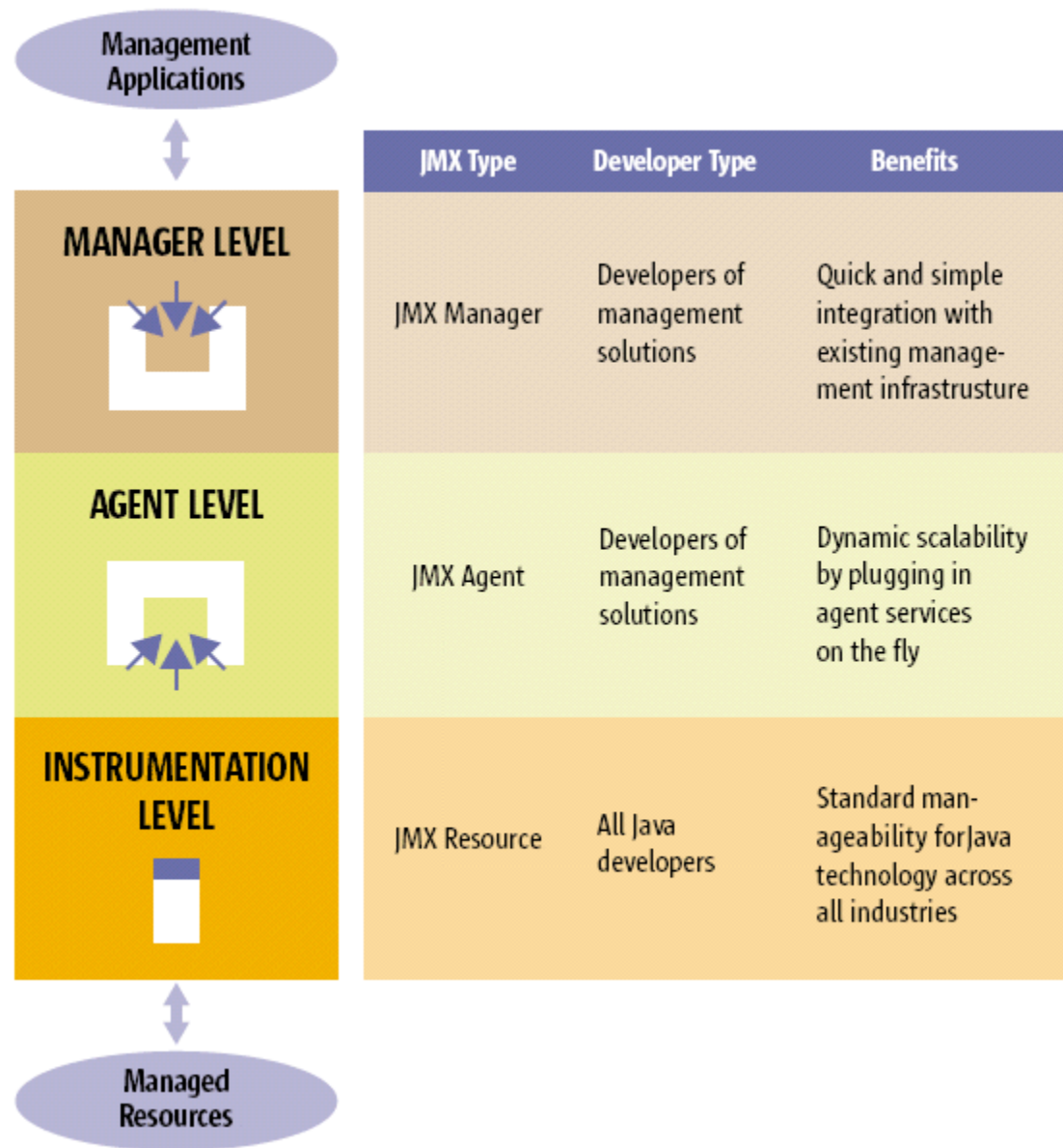


Figure 1: JMX Tiered Architecture

As Figure 1 demonstrates, there are three tiers or levels in this architecture: instrumentation, agent, and manager.

Instrumentation Level

This tier contains MBeans and their manageable resources. It provides a specification for implementing JMX technology-manageable resources, which can be an application, service, device, or user. A resource is manageable if it is developed in Java (or provides a Java wrapper) and has been instrumented so that it can be managed by JMX-compliant applications. A resource is instrumented by one or more MBeans that are either standard or dynamic. Standard MBeans are Java objects that conform to certain design patterns (for example, they must have a constructor and setter/getter methods). A dynamic MBean conforms to a specific interface that offers more flexibility at runtime.

The instrumentation of a resource allows it to be manageable at the agent level; however, note that MBeans do not require knowledge of the JMX agent with which they operate. In other words, any JMX-manageable resource can use any JMX agent that offers the services it requires.

The key Components at the instrumentation level are the MBeans, the notification model, and the MBean metadata classes.

- **MBeans:** An MBean is a Java object that implements a specific interface. The management interface of an MBean is represented as: (1) valued attributes that can be accessed; (2) operations that can be invoked; (3) notifications that can be emitted; and (4) the constructors. There are four types of MBeans:
 - **Standard MBeans:** The simplest to design and implement. Their management interface is described by their method names.
 - **Dynamic MBeans:** They implement a specific interface, and they expose their management interfaces at runtime for greatest flexibility.
 - **Open MBean:** Dynamic MBeans that rely on basic data types for universal manageability; they are self-describing for user-friendliness.
 - **Model MBeans:** Dynamic MBeans that are fully configurable and self described at runtime. They provide a generic MBean class with default behavior for dynamic instrumentation of resources.
- **Notification Model:** JMX technology defines a generic notification model based on the Java event model. It lets developers build proactive management solutions. Using notifications, JMX agents and MBeans can send critical information to interested parties such as management applications or other MBeans.
- **MBean Metadata Classes:** These classes contain the structures to describe all components of an MBean's management interface: its attributes, operations, notification, and constructors. For each of these, the metadata include a name, a description and its particular characteristics (for example, an attribute is readable, writeable, or both; for an operation, the signature of its parameter and return types).

Agent Level

This tier contains the JMX agents used to expose the MBeans. It provides a specification for implementing agents, which control the resources and make them available to remote management applications. Agents are usually located on the same machine as the resources they manage, but this is not a requirement. The JMX agent consists of an MBean server and a set of services for handling MBeans. Managers access an agent's MBeans and use the provided services through a protocol adaptor or connector. But note that JMX agents do not require knowledge of the remote management applications that use them.

The main components at the agent level are the MBean Server and Agent Services.

- **MBean Server:** A registry of objects that are exposed to management operations in an agent. Any object registered with the MBean server becomes visible to management applications. However, note that the MBean server only exposes an MBean's management interface and never its direct object reference. Any resources that you want to manage from outside the agent's JVM must be registered as an MBean in the server. The server also provides a standardized interface for accessing MBeans within the same JVM, giving local objects all the benefits of manipulating manageable resources. MBeans can be instantiated and registered by another MBean, the agent itself, or a remote management application through the distributed services. When you register an MBean, you must assign it a unique object name, which is used by the management application to identify the object on which to perform a management operation.
- **Agent Services:** Objects that can perform management operations on the MBeans registered in the MBean server. By including management intelligence in the agent, JMX helps you build more powerful management solutions. The JMX API defines the following Agent Services available in J2SE 5.0:
 - **Dynamic Class loader:** Through the management applet (m-let) service, retrieves and instantiates new classes and native libraries from an arbitrary network location.

- **Monitors:** Observe the numerical or string value of an attribute of several MBeans and can notify other objects of several types of changes in the target.
- **Timers:** Provide a scheduling mechanism based on a one-time alarm-clock notification or on a repeated, periodic notification.
- **The relation service:** Defines associations between MBeans and enforces the cardinality of the relation based on predefined relation types.

Manager (or Distributed Services) Level

This tier contains the components that enable management applications to communicate with JMX agents. It provides the interfaces for implementing JMX managers, and defines the management interfaces and components that operate on agents. Such components provide an interface for a management application to interact with an agent and its JMX manageable resources through a connector, and also expose a management view of a JMX agent and its MBeans by mapping their semantic meaning into the constructs of a data-rich protocol (such as HTML).

JMX comprises a separate package for each tier of the management architecture. The instrumentation tier will be free, and other tiers can be built from public specifications or reference implementations available under Sun Community Source License. Alternatively, you can purchase commercially supported products.

JMX Technology Implementations

The Java 2 Platform, Standard Edition 5.0 (J2SE 5.0) supports JMX 1.2 and JMX Remote API 1.0, which is now the official JMX reference implementation (RI). For developers who are running J2SE 1.4, a JMX RI is also available from Sun Microsystems, and can be downloaded free of charge.

Sun Microsystems also provides the Java Dynamic Management Kit (Java DMK). Java DMK 5.1 is the first commercial implementation of the latest versions of the JMX standards, JMX 1.2 and JMX Remote API 1.0. JMDK is an all-in-one product for building secure, interoperable monitoring and management solutions on the J2SE platform, and it is supported on Solaris, Microsoft Windows, and Linux. Several other commercial and open source implementations are available as well. It is worth noting that Tomcat 5.0 implements the JMX specification.

JMX Technology and J2SE 5.0

J2SE 5.0 has implemented version 1.2 of the JMX specification and version 1.0 of the JMX Remote API (JSR 160) specification. J2SE 5.0 includes significant monitoring and management features, including:

- **JVM instrumentation:** The JVM is instrumented for monitoring and management providing built-in, out-of-the-box management capabilities for local and remote access.
- **Monitoring and Management APIs:** The `java.lang.management` package provides the interface for monitoring and managing the JVM. It provides access to information such as: number of classes loaded and threads running, memory consumption, garbage collection statistics, on-demand deadlock detection, and others.
- **Management tools** such as JConsole, which is a JMX-compliant monitoring tool that comes with J2SE 5.0. It uses JMX instrumentation of the JVM to provide information on performance and resource consumption of applications running on the Java platform.

The core classes for the JMX implementation are provided in the javax.management package. In addition, the java.lang.management package provides the management interface for monitoring and management of the JVM as well as the operating system on which the JVM is running.

To enable the JMX agent and configure its operation using `jconsole`, you must set some specific system properties when you start the JVM. For local access, set the property `com.sun.management.jmxremote` as follows when starting the JVM:

```
prompt> java -Dcom.sun.management.jmxremote AppName
```

And, to enable monitoring and management from remote systems, set the property:

```
com.sun.management.jmxremote.port=portNumber
```

For more information on setting system properties for JMX, please see [Monitoring and Management using JMX](#).

The JMX Technology Programming Model

Using JMX to instrument your applications, services, or devices for manageability is simple. This is because JMX technology shares Java's object model. If you are familiar with Java and its JavaBeans component model, you already know 95% of all you need to know.

As mentioned previously, an MBean is a Java object that follows some standard design patterns and naming conventions. It can represent a device, application, or any resource that needs to be managed. An MBean exposes a management interface or a set of readable and/or writable attributes and a set of invocable operations, along with a self-description. Note that the management interface does not change through the life of an MBean instance.

Sample Application

This simple application manages a resource. You will create a simple standard MBean that exposes a String object and an operation. For more JMX technology examples, please see the [JMX Tutorial](#).

The first step is to develop the MBean interface. In this application, the interface is called `HelloMBean`, which declares three methods: one getter, one setter, and one for saying *hello* as shown in Code Sample 1.

Code Sample 1: HelloMBean.java

```
public interface HelloMBean {
    public void setMessage(String message);
    public String getMessage();
    public void sayHello();
}
```

The next step is to implement the MBean interface. A sample implementation is shown in the following Code Sample.

Code Sample 2: Hello.java

```
public class Hello implements HelloMBean {
    private String message = null;

    public Hello() {
```

```

    message = "Hello there";
}

public Hello(String message) {
    this.message = message;
}

public void setMessage(String message) {
    this.message = message;
}

public String getMessage() {
    return message;
}

public void sayHello() {
    System.out.println(message);
}
}

```

Congratulations! You have created your first MBean. The next step is to test the MBean, by developing a JMX agent in which you register the MBean. A JMX agent is a component in the agent level and acts as a container for the MBean. A sample agent, `SimpleAgent`, is provided in Code Sample 3. This agent performs the following tasks:

1. Gets the platform MBeanServer
2. Registers an instance of the Hello MBean

Code Sample 3: SimpleAgent.java

```

import javax.management.*;
import java.lang.management.*;

public class SimpleAgent {
    private MBeanServer mbs = null;

    public SimpleAgent() {

        // Get the platform MBeanServer
        mbs = ManagementFactory.getPlatformMBeanServer();

        // Unique identification of MBeans
        Hello helloBean = new Hello();
        ObjectName helloName = null;

        try {
            // Uniquely identify the MBeans and register them with the platform MBeanServer
            helloName = new ObjectName("SimpleAgent:name=hellothere");
            mbs.registerMBean(helloBean, helloName);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Utility method: so that the application continues to run
    private static void waitForEnterPressed() {
        try {
            System.out.println("Press  to continue...");
            System.in.read();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String argv[]) {
        SimpleAgent agent = new SimpleAgent();
        System.out.println("SimpleAgent is running...");
        SimpleAgent.waitForEnterPressed();
    }
}

```

}

The `java.lang.management.ManagementFactory` class is a factory class for getting managed beans for the Java platform. In this example, the `getPlatformMBeanServer()` method is used to get the platform MBeanServer, which is the interface for MBean manipulation on the agent side. It contains the methods necessary for the creation, registration, and deletion of MBeans. The MBeanServer is the core component of the JMX agent infrastructure.

To experiment with this application, do the following:

1. Create a directory of your choice (such as `jmx-example`)
2. Copy Code Samples 1, 2, and 3 into that directory
3. Compile all the `.java` files using `javac`
4. Run `SimpleAgent`. In order to use the `jconsole` tool to manage it, you should run the `SimpleAgent` as follows:

```
prompt> java -Dcom.sun.management.jmxremote SimpleAgent
```

Note: The `-Dcom.sun.management.jmxremote` system property creates an RMI connector to the platform MBeanServer. For information on the RMI connector, see the coming section, [Using the RMI Connector](#).

5. Connect to the JMX agent using the `jconsole` tool. Run the `jconsole` tool from the command line. Once you start `jconsole`, it will display the list of local processes to be monitored as shown in Figure 2.

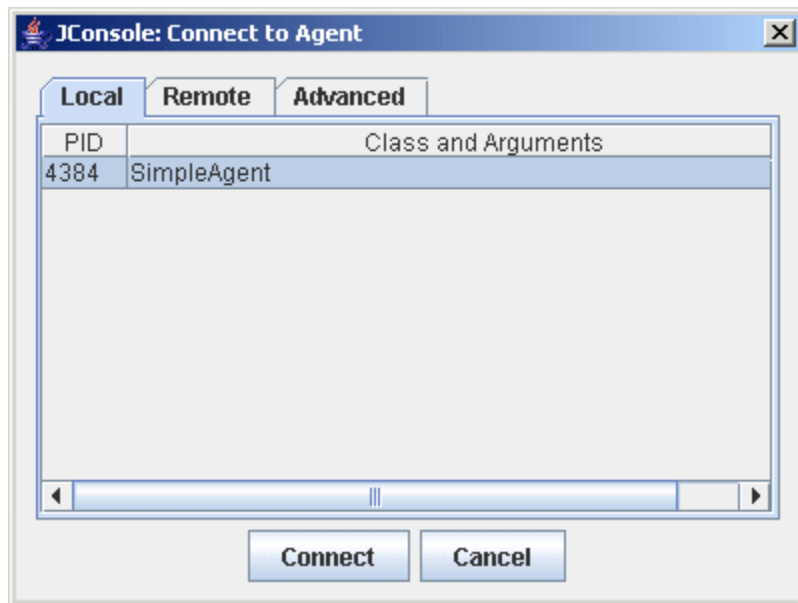


Figure 2: The `jconsole` monitoring tool

6. Now, you can connect to the service. Once connected, select the `MBeans` tab so that you can list the MBeans and manage them as shown in Figure 3.

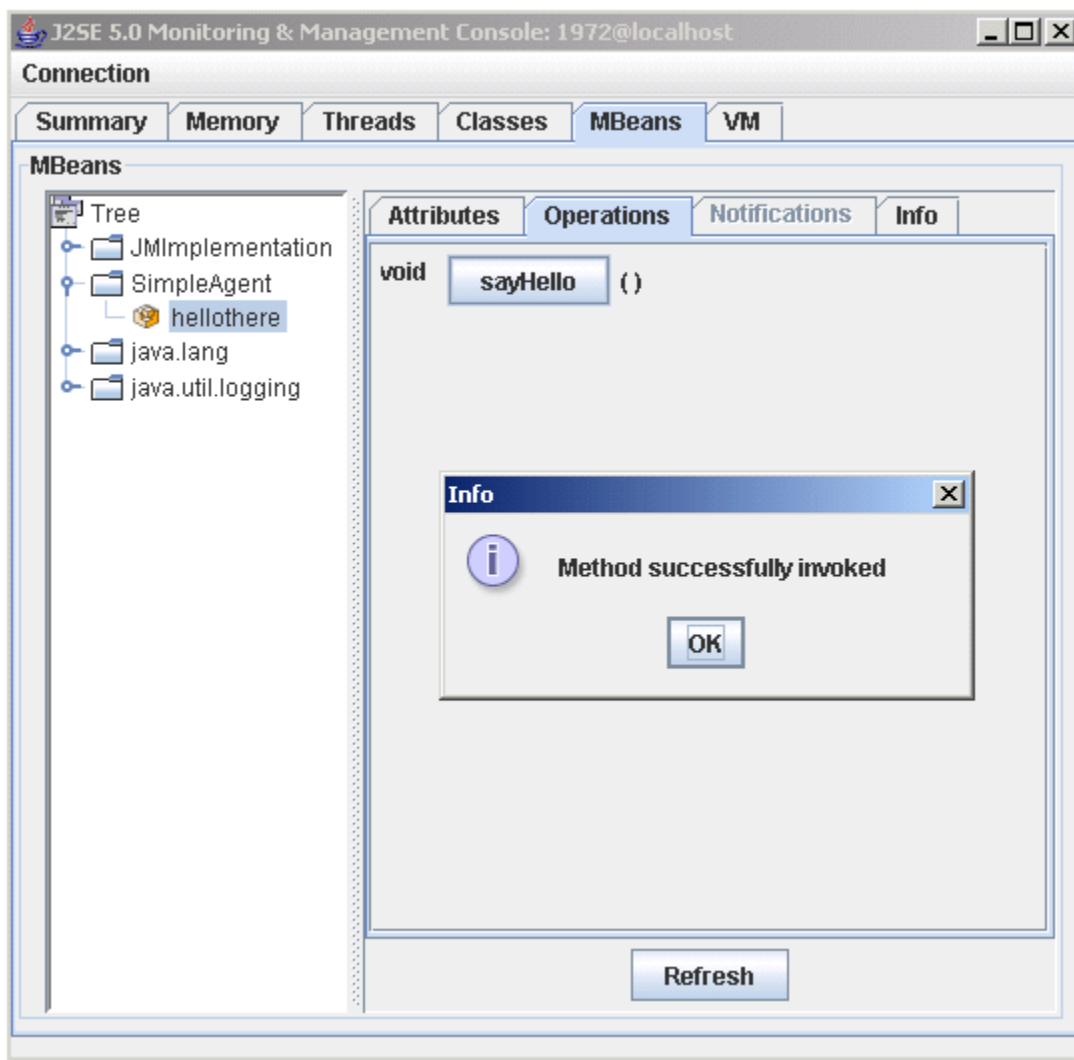


Figure 3: The MBeans tab of jconsole

It is worth noting that local monitoring with `jconsole` is useful for development and prototyping. It is not recommended that `jconsole` be used locally for production environments as it consumes significant system resources. `jconsole` should be used on a remote system from the platform being monitored. For more information on `jconsole` and sample screenshots of what it provides, please see [Using JConsole](#).

JMX Technology for Remote Management

The MBean server relies on protocol adaptors and connectors to make a JMX agent accessible from management applications outside the agent's JVM. Adaptors provide a view through a specific protocol for all MBeans registered in the MBean server (for example, an HTML adaptor could display an MBean in a Web browser). Connectors provide a manager-side interface that handles the communication between manager and JMX agent. When a remote management application uses this interface, it can connect to a JMX agent transparently through the network regardless of the protocol.

JMX technology exports JMX API instrumentation to remote applications by using Remote Method Invocation (RMI). It also defines an optional

protocol based directly on TCP sockets called the JMX Messaging Protocol (JMXMP). However, note that this protocol is not supported in J2SE 5.0.

The JMX Remote API 1.0 (JSR 160) specification describes how to advertise and find JMX agents using existing discovery and lookup infrastructures. In other words, the specification does not define its own discovery and lookup service. Using existing discovery and lookup services is optional; you can encode the address of your JMX API agents in the form of URLs and make these URLs available to the manager.

Using the RMI Connector

Code Sample 4 shows an example of using the RMI connector, which is a slight modification of Code Sample 3.

Code Sample 4: SimpleAgent.java

```
import javax.management.*;
import java.lang.management.*;
import javax.management.remote.*;

public class SimpleAgent {
    private MBeanServer mbs = null;

    public SimpleAgent() {

        // Get the platform MBeanServer
        mbs = ManagementFactory.getPlatformMBeanServer();

        // Unique identification of MBeans
        Hello helloBean = new Hello();
        ObjectName helloName = null;

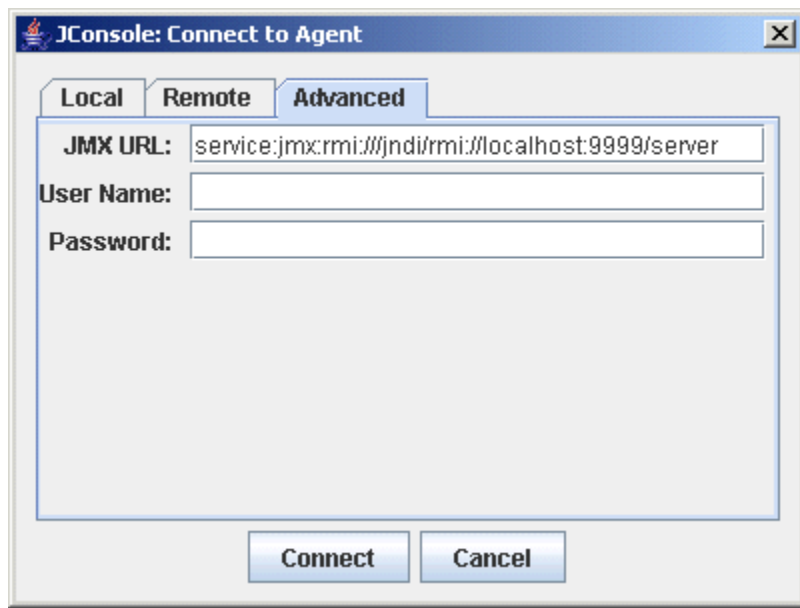
        try {
            // Uniquely identify the MBeans and register them with the MBeanServer
            helloName = new ObjectName("SimpleAgent:name=hellothere");
            mbs.registerMBean(helloBean, helloName);

            // Create an RMI connector and start it
            JMXServiceURL url = new JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:9999/server");
            JMXConnectorServer cs = JMXConnectorServerFactory.newJMXConnectorServer(url, null, mbs);
            cs.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String argv[]) {
        SimpleAgent agent = new SimpleAgent();
        System.out.println("SimpleAgent is running...");
    }
}
```

To run this application, do the following:

1. If you wish to run this application on J2SE 1.4, you need to change the line `mbs = ManagementFactory.getPlatformMBeanServer();` to `mbs = MBeanServerFactory.createMBeanServer("SimpleAgent");`
2. Compile the revised `SimpleAgent.java` from Code Sample 4.
3. Start the `rmiregistry` on port 9999 (**prompt>** `rmiregistry 9999`).
4. Run `SimpleAgent` (**prompt>** `java SimpleAgent`).
5. While `SimpleAgent` is running, run `jconsole` and then use the `Advanced` tab and enter the information as shown in the following figure.



The image shows a Java Swing dialog box titled "JConsole: Connect to Agent". It has three tabs: "Local", "Remote", and "Advanced". The "Advanced" tab is currently selected. Inside the dialog, there are three text input fields: "JMX URL:" containing the text "service:jmx:rmi:///jndi/rmi://localhost:9999/server", "User Name:" which is empty, and "Password:" which is empty. At the bottom of the dialog, there are two buttons: "Connect" and "Cancel".

Figure 4: Configuring jconsole for remote management

6. Once you are connected, you can browse the MBeans and manage them using the `MBeans` tab as shown in Figure 5.

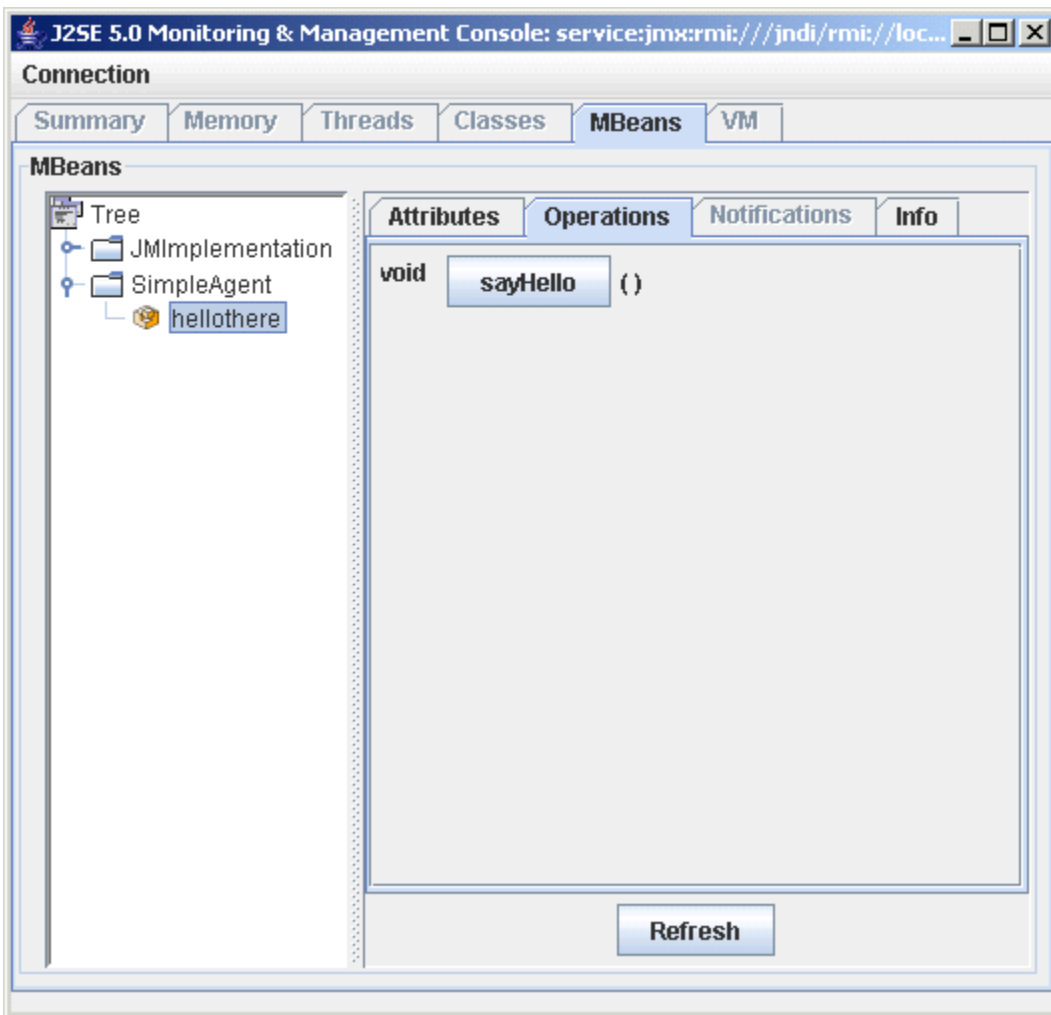


Figure 5: Browsing and managing remote MBeans

Using the HTML Adaptor

An HTML adaptor is provided by Sun Microsystems and comes with the [JMX Reference Implementation](#) (it doesn't come with J2SE 5.0). The HTML adaptor lets you manage MBeans using a web browser. The adaptor enables you to interact with the agent to view registered MBeans and their attributes. In other words, this adaptor provides a simple yet powerful management tool that lets you:

- View the readable MBean attributes
- Update writable attributes
- Invoke methods

Code Sample 5 shows the revised `SimpleAgent` that uses the HTML adaptor.

Code Sample 5: SimpleAgent.java

```
import javax.management.*;
```

```

import java.lang.management.*;
import com.sun.jdmk.comm.HtmlAdaptorServer;

public class SimpleAgent {
    private MBeanServer mbs = null;

    public SimpleAgent() {

        // Create an MBeanServer and HTML adaptor (J2SE 1.4)
        mbs = ManagementFactory.getPlatformMBeanServer();
        HtmlAdaptorServer adapter = new HtmlAdaptorServer();

        // Unique identification of MBeans
        Hello helloBean = new Hello();
        ObjectName adapterName = null;
        ObjectName helloName = null;

        try {
            // Uniquely identify the MBeans and register them with the MBeanServer
            helloName = new ObjectName("SimpleAgent:name=hellothere");
            mbs.registerMBean(helloBean, helloName);
            // Register and start the HTML adaptor
            adapterName = new ObjectName("SimpleAgent:name=htmladapter,port=8000");
            adapter.setPort(8000);
            mbs.registerMBean(adapter, adapterName);
            adapter.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String argv[]) {
        SimpleAgent agent = new SimpleAgent();
        System.out.println("SimpleAgent is running...");
    }
}

```

To experiment with this application, do the following:

1. Download the [JMX Reference Implementation](#). This is needed only because `SimpleAgent` (in Code Sample 5) uses the HTML Adaptor, which does not come with the J2SE 5.0 implementation of JMX 1.2. From the JMX implementation, you only need the `jmxtools.jar` file.
2. If you wish to run this application on J2SE 1.4, you need to change the line `mbs = ManagementFactory.getPlatformMBeanServer();` to `mbs = MBeanServerFactory.createMBeanServer("SimpleAgent");`
3. Compile all the .java files (Code Samples 1, 2, and 5). Make sure you include the `jmxtools.jar` in your classpath.
4. Run `SimpleAgent`. Make sure you include the `jmxtools.jar` in your classpath. `SimpleAgent` will run as a server.
5. Connect to the JMX agent by entering the URL `http://localhost:8000` in your web browser. Once connected to the JMX agent, the HTML adaptor provides three views:
 - Agent view: Provides a summary of the MBeans contained within the agent. You can filter the MBean list to provide refined views.

Agent View

[JDMK5.1_r01]

Filter by object name:

This agent is registered on the domain ***SimpleAgent***.
This page contains **3** MBean(s).

Admin

List of registered MBeans by domain:

- **JMImplementation**
 - [type=MBeanServerDelegate](#)
- **SimpleAgent**
 - [name=hellothere](#)
 - [name=htmladapter.port=8000](#)

Figure 6: HTML adaptor agent view

- MBean view: Provides details about a specific MBean. Here you can set and get MBean attributes and invoke methods as shown in Figure 7.

- **MBean Name:** SimpleAgent:name=hellothere
- **MBean Java Class:** Hello

Reload Period in seconds:

 [Back to Agent View](#)**MBean description:**

Information on the management interface of the MBean

List of MBean attributes:

Name	Type	Access	Value
Message	java.lang.String	RW	<input type="text" value="Hello there"/>

List of MBean operations:[Description of sayHello](#)void

Figure 7: HTML adaptor MBean view

- Admin view: Allows you to register new MBeans on the agent as shown in Figure 8.

Agent Administration[Back to Agent View](#)

Specify the object name and java class of the MBean to add, delete or view the constructors of:
(Optionally provide a class loader name for loading the specified class.)

Domain:	<input type="text" value="SimpleAgent"/>
Keys:	<input type="text"/>
Java Class:	<input type="text"/>
Class Loader:	<input type="text"/>

Action:

JMX Technology Related Specifications

The various JSRs that related to the JMX specification follow:

- JSR 3: [JMX 1.2](#)
- JSR 77: [J2EE Management](#)
- JSR 160: [JMX Remote API 1.0](#)
- JSR 174: [Monitoring and Management Specification for the JVM](#)
- JSR 255: [JMX 2.0](#)
- JSR 262: [Web Services Connector for JMX Agents](#)

Summary

JMX technology provides a component-based architecture for developing solutions to monitor and manage your applications, services, and resources. JMX technology is *the* way to instrument any application or service that was built using Java technology. Hence, the Java platform now provides excellent facilities for creating and managing applications and services. JMX technology should be used for any application and service that benefit from being manageable, as this will increase their value to vendors and clients, by making them easier to install, configure, and maintain. The current version of JMX is 1.2, and the [JMX 2.0](#) update was launched in September 2004. It will update the JMX and JMX Remote APIs to improve existing interfaces, mainly with respect to ease of use.

This article provided a fast track introduction and tutorial to the JMX architecture and its programming model. The sample code provided demonstrates how easy it is to get started developing management and monitoring solutions using JMX technology. The J2SE 5.0 implements the JMX specification—if you use J2SE 5.0, you are ready to start developing using JMX technology.

Related Information

- [Java Management Extensions \(JMX\) Technology](#)
- [Companies and products that use JMX technology](#)
- [J2SE 5.0 JMX Tutorial](#)
- [Monitoring and management in J2SE 5.0](#)
- [J2SE 5.0 JMX Guide](#)
- [JMX Reference Implementation](#)
- [Sun's Java Dynamic Management Kit \(Java DMK\)](#)
- [JMX Marketplace](#)
- [Using JConsole to Monitor Java Applications](#)

Acknowledgments

Special thanks to Mandy Chung and Eamonn McManus of Sun Microsystems, whose feedback helped me improve this article.

Note: *The terms Java Virtual Machine and JVM mean a Virtual Machine for the Java platform.*

Rate and Review

Tell us what you think of the content of this page.

Excellent Good Fair Poor

Comments:

Your email address (no reply is possible without an address):

[Sun Privacy Policy](#)

Note: We are not able to respond to all submitted comments.

Oracle is reviewing the Sun product roadmap and will provide guidance to customers in accordance with Oracle's standard product communication policies. Any resulting features and timing of release of such features as determined by Oracle's review of roadmaps, are at the sole discretion of Oracle. All product roadmap information, whether communicated by Sun Microsystems or by Oracle, does not represent a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. It is intended for information purposes only, and may not be incorporated into any contract.



[About Sun](#) | [About This Site](#) | [Newsletters](#) | [Contact Us](#) | [Employment](#) | [How to Buy](#) | [Licensing](#) | [Terms of Use](#) | [Privacy](#) | [Trademarks](#)

© 2010, Oracle Corporation and/or its affiliates

A Sun Developer Network Site

Unless otherwise licensed, code in all technical manuals herein (including articles, FAQs, samples) is provided under this [License](#).



[Sun Developer RSS Feeds](#)