AppMind Software AB
Linnégatan 87, Box 24 131, S-104 51 Stockholm
Tel: +46 (0) 8 723 27 00
info@appmind.com

AppMind™

A Tutorial

Date: 2002-08-26

AppMind™ and JMX

Executive Summary

This document is an introduction to the JMX part of the AppMind product. It uses the JMS tutorial provided by SUN (see [3]) as a base for describing how to integrate a program with an Enterprise Management System (EMS) by using the standard JMX API.

It is recommended that you have some basic knowledge of the AppMind product. Please read [1] and [2] before starting this tutorial. Furthermore, it is recommended that you are familiar with the JMX architecture. For information about the JMX specification, refer to [4] and [5].

References

[1] "AMT Java API Getting Started – Developer Guide", 0007002

[2] "Go PATROL – Integration Guide", 0105002

[3] Java Message Service Tutorial, http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/copyright.html

[4] Java Management Extensions (JMX) 1.5 Specification JSR 160, http://jcp.org/jsr/detail/160.jsp

[5] Java Management Extensions, http://java.sun.com/products/JavaManagement/

Table of Contents

General 2

Instrument a simple JMS client application 2
Configure your environment 2
Run the JMS example 3
Instrument the JMS example 3
Run the instrumented JMS example 4
Integrate the JMS example with the PATROL framework 7
Add expert advice information to the message 12

Appendix A: JMXTextListener.java 15

Appendix B: The sample file appmind.properties 18

Appendix C: The MMF example 19

General

Java Management Extensions (JMX) is a standard-based API that provides a framework for adding management to user applications. The components of a Java application that uses JMX will have a standard way of exposing some of their functionality for management by using Management Beans, or MBeans. A JMX Agent enables manageability of these functionalities, allowing a system administrator to manage the components of the application. For more information about the architecture of the JMX framework, please refer to [4].

AppMind supports the notification part of the JMX API by providing an adaptor (`com.appmind.jmx.AppMindAdaptor`) that listens for JMX notifications broadcasted by MBeans. The adaptor converts each notification into an ordinary AppMind log message. Once the log message has entered the AppMind framework, it will be treated as any other log message within the AppMind framework. For example, error messages are normally forwarded as events to the preferred EMS.

This tutorial assumes a Windows NT/2000 environment. However, the JMX adaptor is available on all other officially supported platforms.

Instrument a simple JMS client application

Configure your environment

Before you start this tutorial, you need to install the following programs (preferably in the same order as stated below):

1. Java 2 Platform, Standard Edition 1.3.1 (http://java.sun.com/j2se/1.3/download.html)

2. Java 2 Platform, Enterprise Edition 1.3.1 (http://java.sun.com/j2ee/sdk_1.3)

3. JMX Instrumentation Reference Implementation 1.0, binary version (http://java.sun.com/products/JavaManagement/download.html)

4. BMC PATROL Agent & Console, version 3.3 or later

5. AppMind Development Kit, version X.Y or later

6. AppMind Go PATROL Kit, version X.Y or later

Run the JMS example

Please read the tutorial available at http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/client.html#1025458 and run the JMS example titled "A Simple Publish/Subscribe Example" (http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/client.html#1025458).

Instrument the JMS example

Copy the files `JMXTextListener.java` and `JMXTextListenerMBean.java` to the same local directory as the other JMS example files.

The instrumentation code of the JMX example is located in the file `JMXTextListener.java`. `JMXTextListener` is a `TextListener` that registers itself in the MBean server and broadcast each incoming message to all registered listeners (in this case, the AppMind adaptor). The complete source code of `JMXTextListener` is listed in appendix A.

The interface `JMXTextListenerMBean` exposes the management interface of the `JMXTextListener`. We do not have any methods or properties to expose and the interface is therefore empty.

Open the file `SimpleTopicSubscriber.java` in a text editor, go to line 131 and replace the text:

```
  topicListener = new TextListener();
```

with

```
topicListener = new JMXTextListener();
```

Recompile the programs and the instrumented message listener class:

```
javac TextListener.java

javac JMXTextListener.java

javac JMXTextListenerMBean.java

javac SimpleTopicPublisher.java

javac SimpleTopicSubscriber.java
```

Run the instrumented JMS example

The AppMind adaptor reads from a properties file during startup. The properties file specifies the name and origin of the instrumented process. The properties file also specifies a number of keywords for each severity level, which is used by the AppMind adaptor to classify the severity level of an incoming message given its notification type. For an example of a properties file, see Appendix B.

You find a sample properties file at `%AMT_ROOT%\bin` called `appmind.properties`. Copy the sample properties file `%AMT_ROOT%\bin\appmind.properties` to the same directory as the JMS example. Open the file in a text editor and change the properties as listed below:

```
com.appmind.jmx.AppMindAdaptor.application=JMX
com.appmind.jmx.AppMindAdaptor.subsystem=XX
com.appmind.jmx.AppMindAdaptor.program=SAMPLE
```

Open a new command prompt and start the J2EE server:

```
j2ee -verbose
```

Open another command prompt and start the subscriber program:

```
java -Djms.properties=%J2EE_HOME%\config\jms_client.properties
SimpleTopicSubscriber MyTopic
```

You should see the following output produced by the subscriber program:

```
Topic name is MyTopic
Java(TM) Message Service 1.0.2 Reference Implementation (build b14)
Properties from appmind.properties has been loaded
0812-154040-I-Process (XX_SAMPLE 0 PID 0xe90 (3728)) successfully
connected to
              MIB (JMX_MIB_) for instance () in slot 7 (version
2.0), log file:
              D:\Program

Files\twobyfour\AppMind\tmp\jmx_xx_sample_vidstige.log, image
              version: v0.0-000 on NT
```

```
0812-154040-I-XX_SAMPLE in slot 7 changed mode from Startup to
Running
To end program, enter Q or q, then <return>
```

Open another command prompt and verify that the program `XX_SAMPLE` is running in the application named `JMX`:

```
am –app JMX .* show process
```

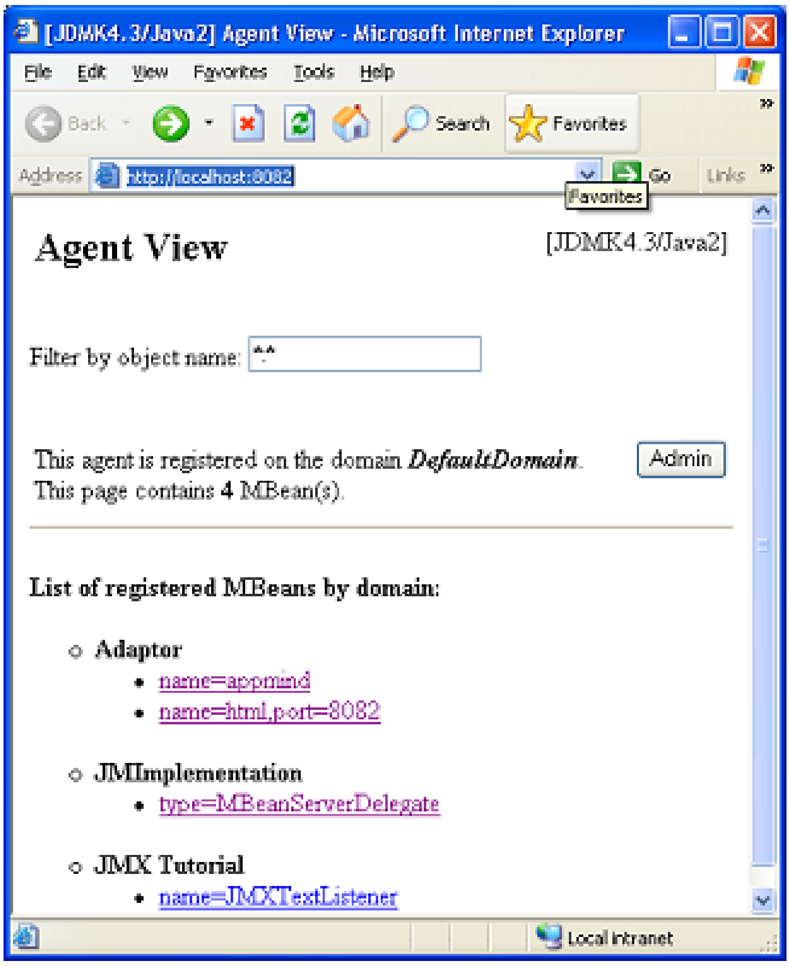
You should see the following output:

```
AMT connected to MIB for application (JMX), instance ()
Slot Process name    id           mode       Log Evt Startup time
---- --------------- ----------   ---------- --- --- ---------------
----
   5 XX_SAMPLE            336   Running     I   E  2002-08-12
14:54:05
```

Which tells us the following:

- The program `XX_SAMPLE` has the process id 336.
- The process is running in process mode "running".
- All messages that are logged with severity "information" or higher are written to the log file.
- All messages that are logged with severity "error" or higher are sent as events to the AppMind framework.

In order to view all registered MBeans in the MBean server, open your favourite web browser and enter the address http://localhost:8082. You should see the following information:


[JDMK4.3/Java2] Agent View - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8082 Go Links

Agent View [JDMK4.3/Java2]

Filter by object name: *:*

This agent is registered on the domain *DefaultDomain*.
This page contains **4** MBean(s).

Admin

List of registered MBeans by domain:

- **Adaptor**
  - name=appmind
  - name=html,port=8082
- **JMImplementation**
  - type=MBeanServerDelegate
- **JMX Tutorial**
  - name=JMXTextListener

Figure 1 The contents of the MBean server seen from a web browser

The adaptor with the object name `appmind` is the AppMind adaptor that listens for JMX notifications emitted by any registered MBean. The adaptor named `html` is the HTML adaptor that allows us to view the contents of the MBean server via a web browser. The HTML adaptor is included in the reference implementation provided by SUN. Finally, the MBean named `JMXTextListener` is the management interface of the `JMXTextListener` object you have implemented in this tutorial.

Run the publisher program:

```
java -Djms.properties=%J2EE_HOME%\config\jms_client.properties
SimpleTopicPublisher MyTopic
```

You should see the following output, where the `JMXTextListener` object produces the second line:

```
Reading message: This is message 1
0812-152736-I-source: JMXExample:name=listener type: info message:
This is message 1
```

Stop the subscriber program:

```
am –app jmx .xx_sample stop process
```

You should see the following output produced by the subscriber program:

```
0814-103345-I-Received command: STOP PROCESS
0814-103345-I-XX_SAMPLE instance 0 terminating as requested
0814-103345-I-XX_SAMPLE instance 0 terminates with status 0
```

You have now successfully instrumented a program and verified that it is successfully integrated with the AppMind framework. The next step is to integrate your instrumented program with the PATROL framework.

Integrate the JMS example with the PATROL framework

Make a copy of the ini-file `%AMT_ROOT%\dat\appmind_patrol_config.ini` and name it `%AMT_ROOT%\dat\jmx_patrol_config.ini`.

Create a node Knowledge Module (KM) for the JMX application by using the AMT Utility:

```
amt –nk JMX_NODE
```

This command creates two files, `JMX_NODE.km` and `JMX_NODE.ctg`. Copy these files to the directory `%PATROL_HOME%\lib\knowledge`.

In order to assure that the JMX KM is loaded each time the PATROL agent is started; add the KM file to the KM list file `%PATROL_HOME%\lib\knowledge\appmind.kml`. The resulting KM list should look like:

```
KM_LIST = {
    "AMT.kml"
    "APPMIND_NODE.km"
    "JMX_NODE.km"
}
```

Restart the PATROL agent:

```
net stop patrolagent

net start patrolagent
```

The PATROL agent will now start an AMT Monitor for the application named JMX. For more details, read the log file `%AMT_ROOT%\tmp\jmx_patrol_<host>.log`, where `<host>` is the name of your local machine.

Verify that the AMT Monitor is up and running:

```
am –app JMX .* sho process
```

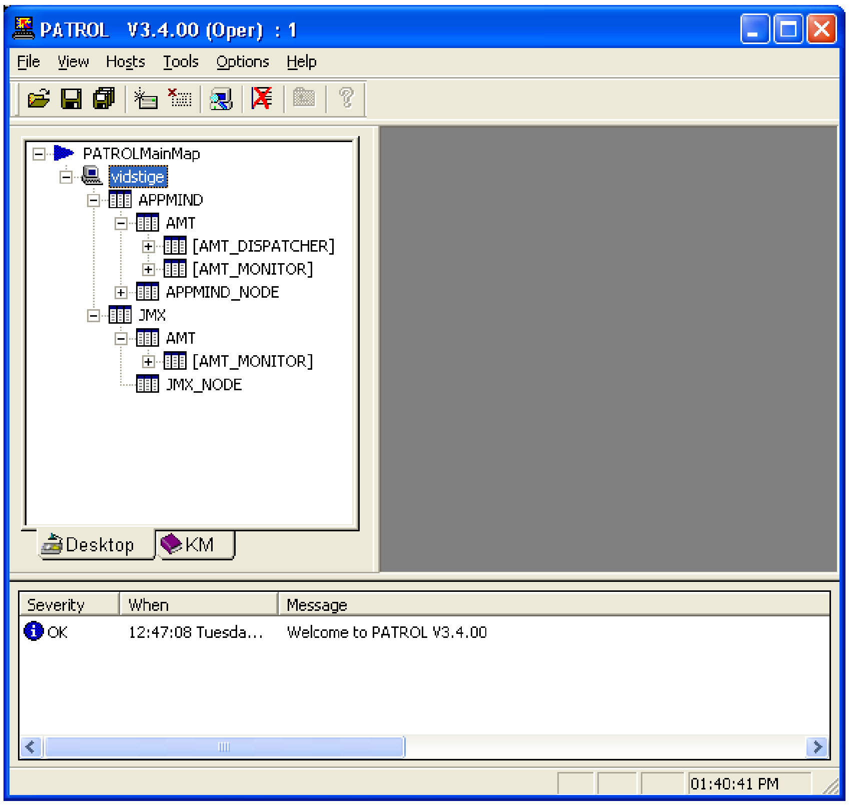
You should see the following output:

```
AMT connected to MIB for application (JMX), instance ()
Slot Process name    id           mode       Log Evt Startup time
---- --------------- ----------   ---------- --- --- ---------------
----
   2 AMT_MONITOR           2508   Server      I   E  2002-08-13
11:01:22
```

Which tells us the following:

- The program `AMT_MONITOR` has the process id 2508.
- The process is running in the process mode "Server". That is, the AMT Monitor is successfully connected to the PATROL agent.
- All messages that are logged with severity "information" or higher are written to the log file.
- All messages that are logged with severity "error" or higher are sent as events to the AppMind framework.

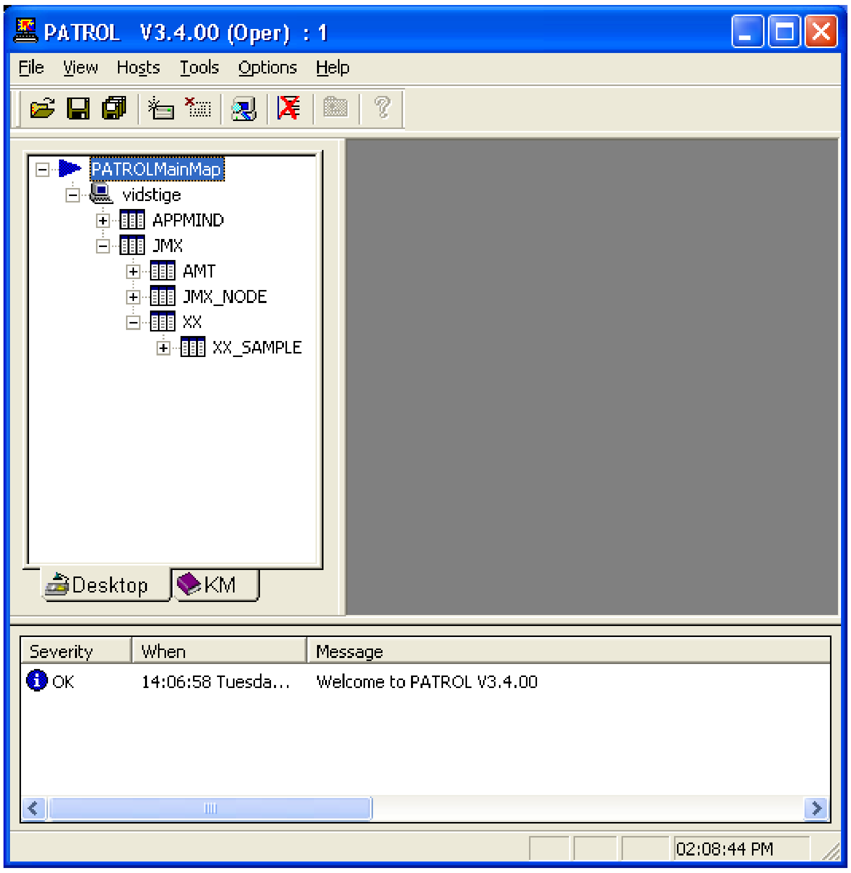
Start the PATROL Operator Console and load the KM file by clicking the menu item `File/Load KM`… If you expand the tree in the main map, you should see the following structure:



```
Severity  When              Message
OK        12:47:08 Tuesda...  Welcome to PATROL V3.4.00
```

Figure 2 The PATROL Console

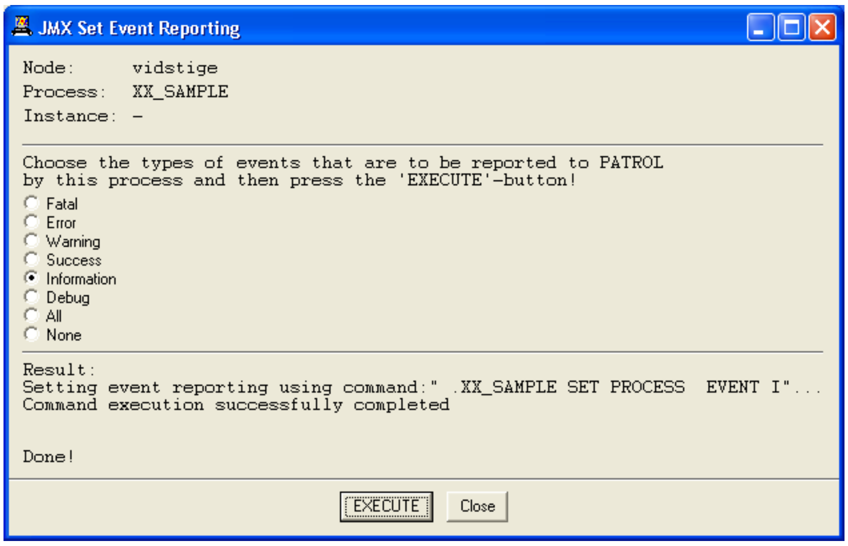
Start the subscriber program again. You should now see the program XX_SAMPLE in the main map as illustrated below:



Severity	When	Message
OK	14:06:58 Tuesda...	Welcome to PATROL V3.4.00

Figure 3 The sample program presented in the main map

Now, change the event severity level for the subscriber program application from “error” to “information” by right clicking on the node `XX_SAMPLE` and then the popup menu item named `KM Commands/Set Event Reporting`…. A dialog should now appear. Select the radio button “Information” and then click on the button “Execute”. The dialog should now look like:



```
Node:     vidstige
Process:  XX_SAMPLE
Instance: -

Choose the types of events that are to be reported to PATROL
by this process and then press the 'EXECUTE'-button!
```

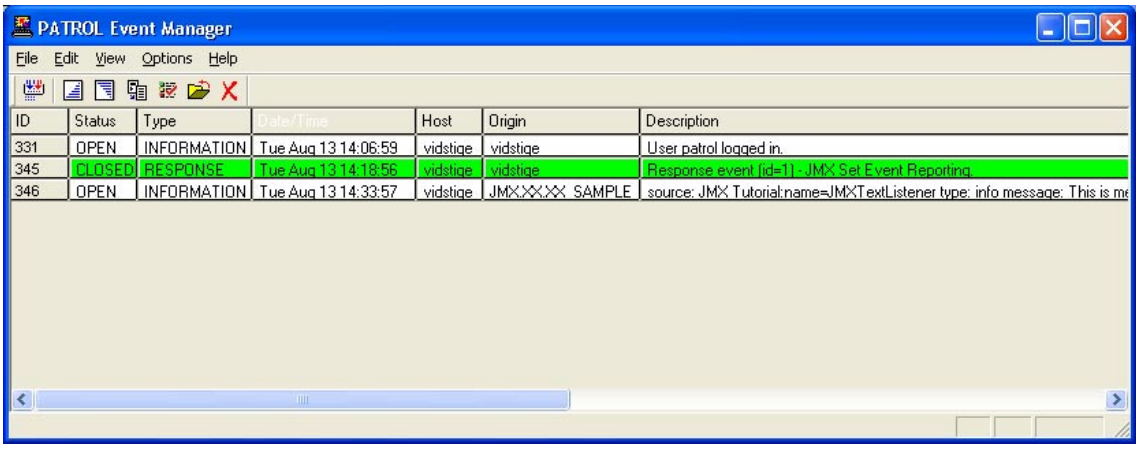
( ) Fatal
( ) Error
( ) Warning
( ) Success
[✓ Information]
( ) Debug
( ) All
( ) None

```
Result:
Setting event reporting using command:" .XX_SAMPLE SET PROCESS  EVENT I"...
Command execution successfully completed


Done!
```

Figure 4 The Set Event Reporting dialog

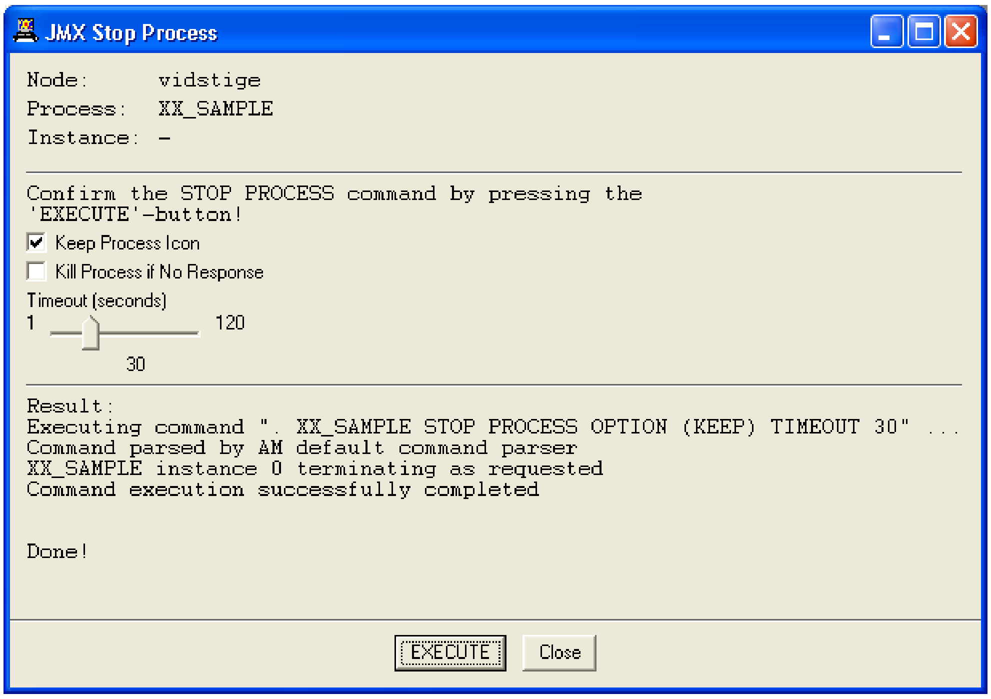
Close the dialog and open the PATROL Event Manager by clicking the menu item `Tools/Event Manager`…. Start the publisher program. The message should appear in the Event Manager window as illustrated below:



ID	Status	Type	Date/Time	Host	Origin	Description
331	OPEN	INFORMATION	Tue Aug 13 14:06:59	vidstige	vidstige	User patrol logged in.
345	CLOSED	RESPONSE	Tue Aug 13 14:18:56	vidstige	vidstige	Response event (id=1) - JMX Set Event Reporting.
346	OPEN	INFORMATION	Tue Aug 13 14:33:57	vidstige	JMX.XX.XX_SAMPLE	source: JMX Tutorial:name=JMXTextListener type: info message: This is me

Figure 5 The message presented as an event in the Event Manager window

Stop the subscriber program by right clicking on the icon `XX_SAMPLE` in the main map and then on the menu item `KM Commands/Stop Process`…. Click on the execute button in the dialog to confirm the stop command. The dialog should display the result as illustrated below:



```
Node:      vidstige
Process:   XX_SAMPLE
Instance:  -

Confirm the STOP PROCESS command by pressing the
'EXECUTE'-button!
[✓] Keep Process Icon
[ ] Kill Process if No Response
Timeout (seconds)
1                 120
        30

Result:
Executing command ". XX_SAMPLE STOP PROCESS OPTION (KEEP) TIMEOUT 30" ...
Command parsed by AM default command parser
XX_SAMPLE instance 0 terminating as requested
Command execution successfuly completed


Done!
```

Figure 6 The Stop Process dialog

You have now successfully integrated your JMS program with PATROL. The next step is to take advantage of the Message Management File (MMF) concept, which allows you to associate a so-called expert advice to a given message.

Add expert advice information to the message

User-defined messages are specified in a Message Management File (MMF). In this file, you describe the meaning of each message, its severity, and any actions the operator must take in order to resolve the problem. See Appendix C for the MMF used in this tutorial.

Copy the file `XXMessages.mmf` to the same local directory as the other JMS example files. Generate the java source code from the MMF:

```
amt -m XXMessages.mmf -java
```

You should see the following output:

```
amt: INFO   : Using the message base 888000 for subsystem XX as
specified in
              file: 'XXMessages.mmf'!
amt: INFO   : Now parsing file 'XXMessages.mmf'...
amt: INFO   : Found 1 messages in file 'XXMessages.mmf'!
amt: INFO   : Creating java file 'XXMessages.java'
amt: INFO   : Creating message definition file 'XXMessages.mdf'
amt: INFO   : Finished with 0 errors and 0 warnings.
amt: INFO   : Summary information per subsystem:

 SUBSYSTEM   INFORMATIONS      WARNINGS        ERRORS
TOTAL
 =========   ============   ===========   ===========
===========
        XX              1             0             0
1

amt: SUCCESS: Task successfully completed!
```

Copy the Message Definition File (MDF) `XXMessages.mdf` to `%AMT_ROOT%\dat`. This file is the binary form of the message definitions and is read by the AppMind framework during startup or manually via the AM Command. Read the new message definitions:

```
am –app JMX refresh messages
```

You should see the following output:

```
0815-151946-I-Message Definitions allocated = 267, reread = 215
0815-151946-I-Subsystems allocated = 30, reread = 3
Command execution successfully completed
```

Open the file `JMXTextListener.java` in a text editor and import the `AppMindNotification` class in the beginning of the file:

```
import com.appmind.jmx.AppMindNotification
```

Furthermore, go to line 80 and replace the text:

```
Notification notification = new Notification("info",
                                             name,
                                             0,
                                             msg.getText());
```

with

```
AppMindNotification notification = new AppMindNotification(null,
                                                            name,
                                                            0);
notification.setMessageId(XXMessages.XX_MSG_RECEIVED);
```

Recompile the files:

```
Javac XXMessages.java

javac TextListener.java

javac JMXTextListener.java

javac JMXTextListenerMBean.java

javac SimpleTopicPublisher.java

javac SimpleTopicSubscriber.java
```

Create a subsystem KM for the JMX application by using the AMT Utility:

```
amt –km JMX_XX.km –m XXMessages.mmf
```

This command creates two files, `JMX_XX.km` and `JMX_XX.ctg`. Copy these files to the directory `%PATROL_HOME%\lib\knowledge`.

In order to assure that the subsystem KM is loaded each time the PATROL agent is started; add the KM file to the KM list file `%PATROL_HOME%\lib\knowledge\appmind.kml`. The resulting KM list should look like:

```
KM_LIST = {
    "AMT.kml"
    "APPMIND_NODE.km"
    "JMX_XX.km"
    "JMX_NODE.km"
}
```

Restart the PATROL agent:

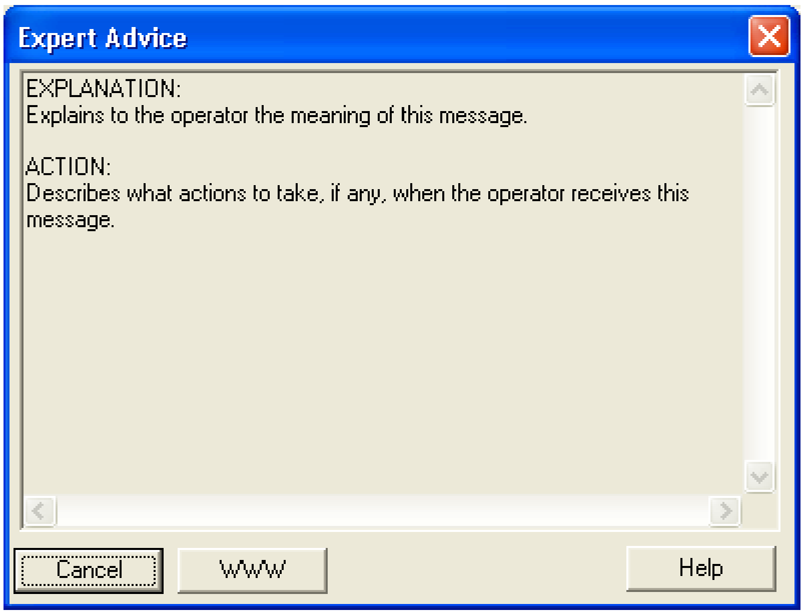
```
net stop patrolagent

net start patrolagent
```

Start the PATROL Operator Console and load the `JMX_XX.km` file by clicking the menu item `File/Load KM`… Assure that the `JMX_XX.km` precedes `JMX_NODE.km` in the list of loaded KM's. If not, unload the `JMX_NODE.km` and then load it again.

Start the subscriber program and wait for the node to appear in the main map. Adjust the event severity level for the process `XX_SAMPLE` as described earlier.

Open the Event Manager window and then start the publisher program. Double-click on the event and then click on the button labeled "Expert Advice". You should now see the same descriptive text you entered in the MMF file.



EXPLANATION:
Explains to the operator the meaning of this message.

ACTION:
Describes what actions to take, if any, when the operator receives this message.

Figure 7 The expert advice is presented in a separate dialog

This concludes the "AppMind and JMX" tutorial.

Appendix A: JMXTextListener.java

```
/**
 * The JMXTextListener class extends the TextLister class by
broadcasting
 * each incoming JMS message to all registered listeners.
 */
import javax.jms.*;

import java.util.ArrayList;

//
// JMX-related imports.
//
import javax.management.ObjectName;
import javax.management.MBeanServer;
import javax.management.MBeanServerFactory;
import javax.management.Notification;
import javax.management.NotificationListener;
import javax.management.NotificationFilter;
import javax.management.NotificationBroadcaster;
import javax.management.NotificationBroadcasterSupport;
import com.sun.jdmk.comm.HtmlAdaptorServer;

public class JMXTextListener
    extends TextListener
    implements MessageListener, JMXTextListenerMBean,
NotificationBroadcaster
{
    public JMXTextListener()
    {
        super();

        /*
         * Start the MBean server and add the HTTP adaptor.
         */
        MBeanServer server = null;
        try
        {
             broadcaster = new NotificationBroadcasterSupport();
             server = MBeanServerFactory.createMBeanServer();
             com.appmind.jmx.AppMindAdaptor adaptor;
             adaptor = new com.appmind.jmx.AppMindAdaptor();
             ObjectName name = new
ObjectName("Adaptor:name=appmind");
             server.registerMBean(adaptor, name);

             HtmlAdaptorServer html = new HtmlAdaptorServer();
             name = new ObjectName("Adaptor:name=html,port=8082");
             server.registerMBean(html, name);
             html.start();
        }
        catch (Exception e)
        {
            System.out.println("Failed to start the MBeanServer: " +
                               e.getMessage());
            return;
        }

        try
        {
            name = new ObjectName("JMX
Tutorial:name=JMXTextListener");
            server.registerMBean(this, name);
        }
        catch (Exception e)
        {
            System.err.println("Failed to register the
JMXTextListener: " +
                               e.toString());
        }
    }

    /**
     * Casts the message to a TextMessage and displays its text.
     *
     * @param message     the incoming message
     */
    public void onMessage(Message message)
    {
        super.onMessage(message);
        try
        {
            if (message instanceof TextMessage)
            {
                TextMessage msg = (TextMessage) message;
                Notification notification = new Notification("info",
                                                              name,
                                                              0,

msg.getText());
                broadcaster.sendNotification(notification);
            }
        }
        catch (Exception e)
        {
             Notification notification = new Notification("error",
                                                           name,
                                                           0,

e.toString());
             broadcaster.sendNotification(notification);
        }
    }

    public void removeNotificationListener(NotificationListener
listener)
        throws javax.management.ListenerNotFoundException
    {
        broadcaster.removeNotificationListener(listener);
    }

    public javax.management.MBeanNotificationInfo[]
getNotificationInfo()
    {
        return broadcaster.getNotificationInfo();
    }

    public void addNotificationListener(NotificationListener
listener,
                                        NotificationFilter filter,
                                        java.lang.Object obj)
        throws java.lang.IllegalArgumentException
    {
        broadcaster.addNotificationListener(listener, filter, obj);
    }
    private ObjectName name;
    private NotificationBroadcasterSupport broadcaster;
}
```

Appendix B: The sample file appmind.properties

```
#
# This is the application the JMX instrumented process is part of.
#
com.appmind.jmx.AppMindAdaptor.application=JMX

#
# The subsystem the JMX instrumented program is part of
#
com.appmind.jmx.AppMindAdaptor.subsystem=XX

#
# The program instance, which could be one of SINGLE, DYNAMIC or an
# integer value less than 999
#
com.appmind.jmx.AppMindAdaptor.program.instance=SINGLE

#
# All possible suffix strings of a notification type that should be
# considered as a warning.
com.appmind.jmx.AppMindAdaptor.warning.types=warning|caution

#
# All possible suffix strings of a notification type that should be
# considered as an informational message.
#
com.appmind.jmx.AppMindAdaptor.info.types=info|success

#
# All possible suffix strings of a notification type that should be
# considered as an error.
#
com.appmind.jmx.AppMindAdaptor.error.types=error|fatal|critical|aler
t|failure

#
# The name of the JMX instrumented program
#
com.appmind.jmx.AppMindAdaptor.program=SAMPLE
```

Appendix C: The MMF example

```
subsystem: XX,888000

code:           MSG_RECEIVED
message:        Reading message:
explanation:    Explains to the operator the meaning of this
message.
action:         Describes what actions to take, if any, when the
operator receives this message.
severity: ERROR
```