Developing Java™ Applications

Solstice Enterprise Manager™ 4.1

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

Part No. 806-7972-10
October 2001, Revision A

Please
Recycle

Adobe PostScript

Contents

Figures

Tables

Code Samples

Preface

The *Developing Java Applications* book explains how to develop Java™ applications using the Java Management Interface (JMI), Alarm, and Topology APIs of Solstice Enterprise Manager (Solstice EM). In addition, it explains how to manage Java Dynamic Management™ Kit (JDMK) agents with Solstice EM. The *Developing Java Applications* book is a companion document to *Java API Reference*, which provides a definitive list of the Java API classes and methods.

Who Should Use This Book

This document is intended for telecommunications application developers and service providers developing Java applications and managing JDMK agents with Solstice EM. No prior experience with Solstice EM is required. However, if you are not familiar with Solstice EM, see Section "Related Books" on page xiv for a listing of books to refer.

Before You Read This Book

Read through the Solstice EM documentation so that you have an understanding of the programming context, because many parts of this book refer to concepts that are covered elsewhere in the Solstice EM documentation.

It is also recommended that you familiarize yourself with JDMK concepts, explained in your JDMK documentation.

How This Book Is Organized

This book contains the following chapters:

Chapter 1 "**Introduction**" is an overview of the Java / MIS architecture.

Chapter 2 "**Using the Java Management Interface API**" explains how to use the Java Management Interface (JMI) API to create general Solstice EM management applications.

Chapter 3 "**Using the Java Alarm API**" details how to create alarm management applications using the Java Alarm API.

Chapter 4 "**Using the Java Topology API**" describes how to create Java topology applications.

Chapter 5 "**Configuring the JDMK Agent/Behavior Service**" explains how to manage JDMK agents with Solstice EM.

Appendix A "**Using the Java Alarm and Topology APIs Together**" provides an example that shows you how to use the Java Alarm and Java Topology APIs together.

Related Books

Following is a list of related books:

- *Java API Reference*
- *Developing C++ Applications*
- *C++ API Reference*
- *Management Information Server (MIS) Guide*
- *Managing Your Network*
- *Customizing Guide*

What Typographic Changes Mean

The following table describes the typographic changes used in this book.

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output.	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output.	`machine_name% su` `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value.	To delete a file, type `rm` *filename*.
AaBbCc123	Book titles, new words or terms, or words to be emphasized.	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P-2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

Accessing Sun Documentation Online

The docs.sun.com[sm] web site enables you to access Sun technical documentation on the Web. You can browse the docs.sun.com archive or search for a specific book title or subject at http://docs.sun.com

Also, you can view the online documentation by pointing your browser to the following URL, file:/opt/SUNWconn/em/docs/SEMDOCHP/index.html

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can send your comments by email to docfeedback@sun.com.

Please include the part number of your document in the subject line of your email.

CHAPTER 1

Introduction

This chapter provides an overview of the Java development tools and associated infrastructure that enables you to create Java applications and manage Java Dynamic Management Kit (JDMK) agents with Solstice Enterprise Manager (Solstice EM).

This chapter covers the following topics:

1.1 Important Terms

The following table lists terms that are used in this book.

TABLE 1-1 Important Terms

Term	Description
MPA	Management Protocol Adaptor. A Solstice EM term used for a daemon process that maps the Common Management Information Protocol (CMIP) to another protocol.
JavaBeans™ component architecture	A Java component specification that requires individual Java classes to conform to strict design specifications thereby allowing plug and play with other classes that conform to the same specifications.
JDMK	Java Dynamic Management Kit. A set of Java classes, Java interfaces, and tools that simplify the development of management services.

1.2 Architectural Overview

Before starting to develop Java applications with Solstice EM, it is important for you to understand how the Java components required to do the job relate to the Management Information Server (MIS) architecture; as illustrated in the following figure.

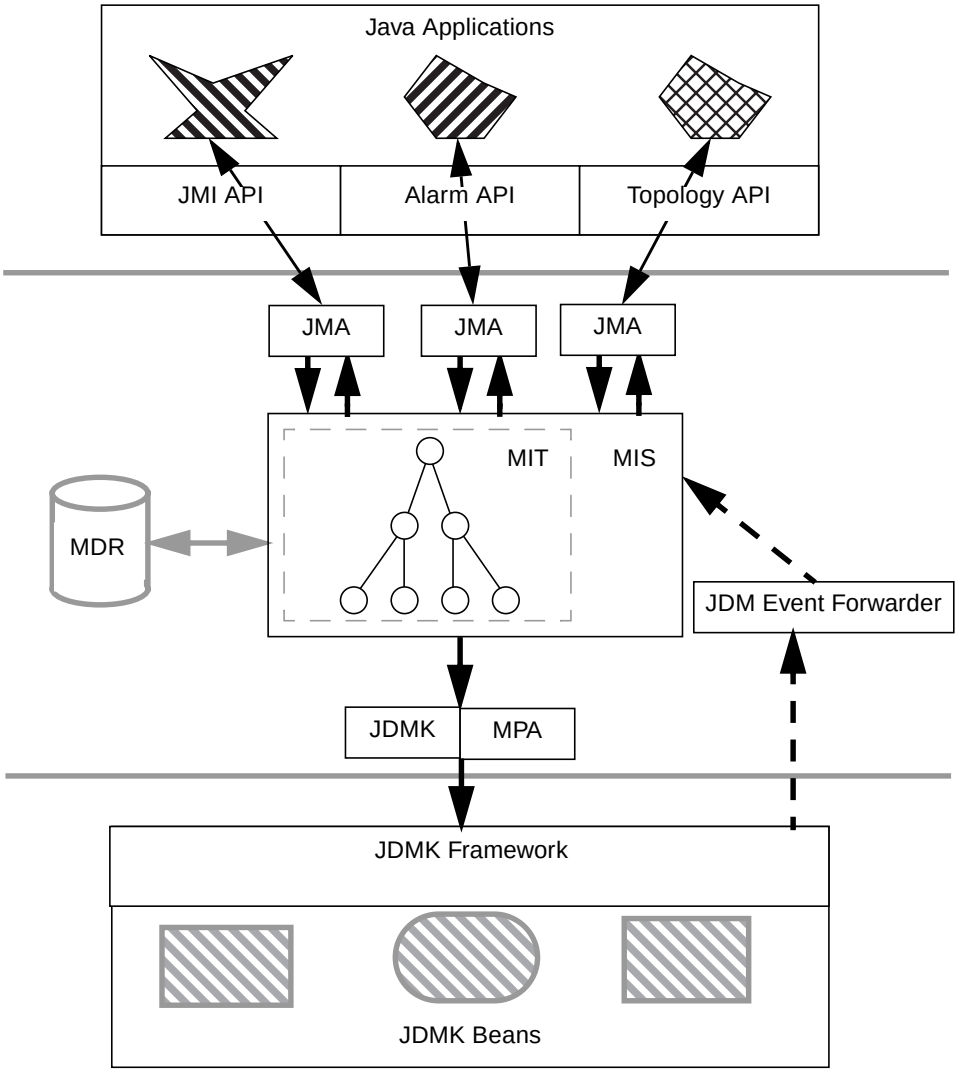

FIGURE 1-1 Architectural Overview

As you can see in FIGURE 1-1, there are two main components for developing Java solutions with Solstice EM:

- The Java API environment consisting of the JMI API, Alarm API, and Topology API

  Allows you to develop Java applications that can interact with the MIS transparently.

- JDMK

  Allows you to build agents.

These two development components are supported by other “adaptor” components that allow Java and C++ Solstice EM applications, JDMK agents, and other types of agents to communicate seamlessly through the MIS.

Note – The Java APIs and associated tools of Solstice EM require the Java 2 SDK. Earlier versions of the Java Development Kit are not compatible with Solstice EM.

1.3 Java APIs

The JMI API, Alarm API, and Topology APIs allow you to develop low-cost, multi-platform Java applications to help manage your network. These applications take full advantage of the advanced distributed management services provided by Solstice EM.

These APIs follow the thin class paradigm. They delegate the bulk of their implementation to a more powerful server. Class state and cache data are located on the server to minimize memory usage by the client.

In addition, it is totally transparent to the user whether a method of a class in these APIs is executed locally or remotely. As a result, you can continue to use these classes in the same manner as any other programming language, like C++.

For example, if you instantiate the `Platform` class, some of the methods in this class would be executed locally, while other methods would be executed on a corresponding remote class or object present on a JMA server.

The Java APIs and associated infrastructure are optimized for efficiency and performance based on the application processing requirements and available resources. For example:

- Memory or CPU intensive operations occur on the server transparently.
- Both the Topology and Alarm APIs provide methods to enable the batch-loading of records, which improves the responsiveness of the client.
- Class state and cache data are located on the server to minimize memory usage by the client.

1.4 JDMK MPA

The JDMK Management Protocol Adapter (MPA) maps Common Management Information Service (CMIS) requests into JavaBeans JDMK calls. The JDMK MPA allows Java applications to access JavaBeans components through JDMK as GDMO objects. However, the JDMK MPA does *not* allow Java objects to make CMIS requests back into the MIS. You can only make such requests through the JMI API. A JDMK agent can send M-Events to the MIS through the JDMK to CMIS Event listener, which routes events through the JDM Event component. (See FIGURE 1-1.)

1.5 JDMK to CMIS Event Forwarder

The JDMK to CMIS Event listener is a daemon process that converts JDMK JavaBean events to Solstice EM CMIS notifications. JDMK agents can send events through the JDMK event mechanism that translates them into Solstice EM CMIS notifications. The type of notification generated is handled transparently as part of the JavaBean to GDMO tool. Any JavaBean that implements the `addXXXListener` and `removeXXXListener` classes causes the JavaBean to GDMO tool to generate an `XXX` notification type.

1.6 JMA

The Java Management Adapter (JMA) provides the framework for the thin client / fat server model. JMA is not exposed to end users or developers. It is a transparent component lying between services such as the JMI and the MIS.

JMA provides the infrastructure for services such as JMI API, Topology API, and Alarm API to communicate seamlessly with the MIS. It is responsible for the scheduling and synchronization of all PMI calls made by each Java API. It provides an event handling mechanism, which allows clients to register their own events and servers to forward the events to the clients.

CHAPTER 2

Using the Java Management Interface API

The JMI API provides a set of classes and methods that allow effective access to the Solstice Enterprise Manager (Solstice EM) Management Information Server (MIS) without requiring detailed specification of the underlying MIS or mechanism. For most applications, the high-level usage of the JMI API is sufficient for all interaction with the Solstice EM MIS.

This chapter describes the following topics:

These topics are arranged, as far as possible, to reflect the order which you need to follow when creating Java management applications. For example, before handling events, you should have defined a `Platform` object, instantiated handlers, and registered event types. This sequence will become clear as you go through this chapter.

Also note, the code samples used in this chapter are based on the sample JMI application that is provided in Section 2.7 “Sample JMI Application” on page 2-14.

2.1 Overview

The JMI API enables you to develop client applications that perform the following operations:

- Accessing information about managed objects inside an MIS.
- Performing object management tasks that are not supported by the existing Solstice EM components.

For example:

- You may want to present information in a fashion that is not possible using the Solstice EM Viewer or the Alarms window. The presentation of information could include specialized presentation window, GUI-based device front ends, or terminal output.
- You may want to manipulate information in a manner that is not possible using Solstice EM subcomponents. The manipulation of information could include summarization of data and specialized gathering and processing of data.

Because of its generic nature, the JMI API could be used to provide the same functionality that other specialized Java APIs provide, such as the Java Alarm API and the Java Topology API.

2.1.1 Java Management Tasks

The JMI API allows you to perform the following tasks:

- Initializing objects that simplify connecting with the Solstice EM MIS;
- Accessing event notification, subscription, and propagation services;
- Accessing object instance and object class information; and
- Representing and managing the different types of relationships.

These tasks could be performed using the following main classes of the JMI API:

- `MOHandle` class
- `MOHCollectionByRule` and `MOHCollectionEnum` classes (subclasses of the `MOHCollection` class)
- `EventReportListener` interface

The interaction between JMI API classes and the MIS is illustrated in the following figure.

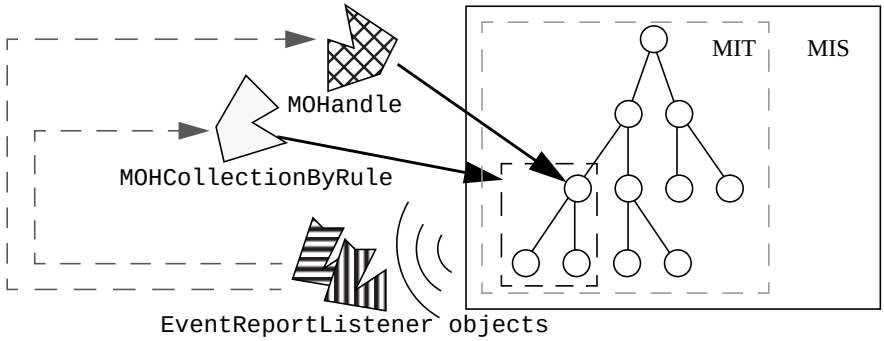

FIGURE 2-1 Interaction Between JMI API Classes and the MIS

The `MOHandle`, `MOHCollectionByRule`, and `MOHCollectionEnum` classes provide the necessary functionality for managing objects over the network, whether individually (`MOHandle`) or as a collection (`MOHCollectionByRule` and `MOHCollectionEnum`).

The `EventReportListener` interface allows you to define event handler classes and their handler methods that will be invoked upon event delivery.

2.1.2 Java Management Task Flow

When creating Java management applications, keep in mind the task flow as illustrated in the following figure.

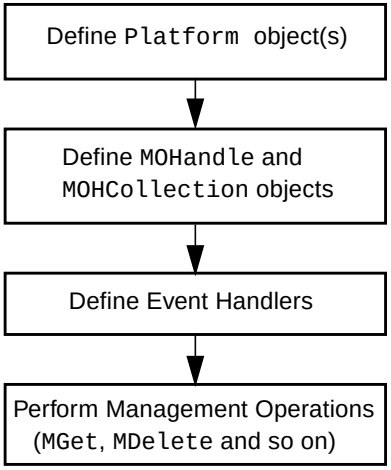

FIGURE 2-2 Task Flow in a Java Management Application

2.2 Instantiating the `Platform` Class

An instance of the `Platform` class represents an actual or potential connection to a Solstice EM MIS, along with all the implied semantics of the particular MIS. You use a `Platform` object to gain access to an MIS.

The first step in creating a JMI application is to define the server instance that allows your application to talk to the MIS. You can do this by creating an instance of the `Platform` class. See FIGURE 2-3.

Note – You can instantiate multiple `Platform` objects in your application so that it can connect to multiple MIS environments.

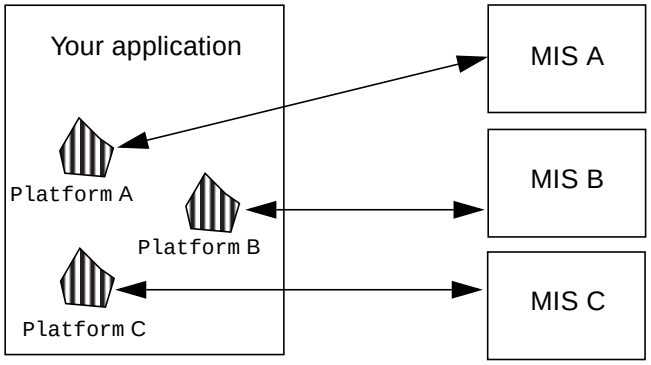

FIGURE 2-3 `Platform` Objects

The `Platform` object you create becomes the key that the `MOHandle` and `MOHCollection` classes you instantiate later will use to access management information.

In addition, the `Platform` object allows you to register Callbacks (see Section 2.4 “Registering Event Listeners” on page 2-10) and allows you to perform access control.

For more information about the `Platform` class, refer to Chapter 2 “Java PMI API” in *Java API Reference.*

To create a `Platform` object, you need to provide the following four parameters:

- Host (on which the JMA is running)
- MIS name
- User name
- Password

Following is a code segment that creates a `Platform` object:

```
plat = new Platform(host, mis, user, passwd);
```

2.3 Defining Local Representation of Managed Objects

When you create a JMI application, you must define a local representation of the network's Managed Object Instances (MOI) that your application will manage. You can perform this task by instantiating the following JMI API classes:

- `MOHandle`: Allows you to represent a single MOI (physical or conceptual).
- `MOHCollectionByRule` and `MOHCollectionEnum`: Allow you to represent groups of MOIs (physical or conceptual).

2.3.1 Instantiating the `MOHandle` Class

An instance of the `MOHandle` class is the local representation of an actual or potential MOI. Typically, an MOI is a managed object that represents a physical resource: a host, server, router, subnet (that is, the representation of a physical device), or a conceptual entity (a line, a queue, or some other aspect of network operation that can be represented as a managed object). See the following figure.

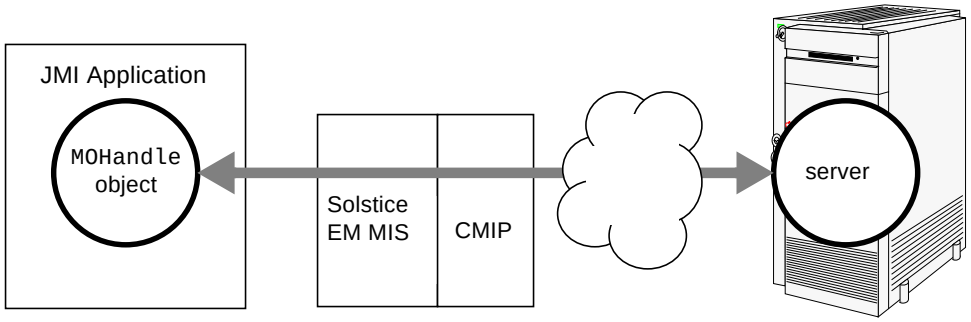

FIGURE 2-4 `MOHandle` Object

Think of the `MOHandle` object as the object itself, even though the actual object is across the network, or in the MIS. `MOHandle` objects give you access to the object's methods and attributes.

To instantiate the `MOHandle` class, you need to provide the following three parameters:

- `dn`: `DistinguishedName` for the MOI.

- `Class name`: The name of the managed object class of the MOI specified in the `dn` parameter. The name must be the same as the name defined in the GDMO specification of the managed object class.
- `Platform`: The `Platform` object you instantiated earlier in the application.

The following code example creates an `MOHandle` instance:

CODE EXAMPLE 2-1 Instantiating the `MOHandle` Class

```
    String dn = new String("topoTypeDBId=NULL/
topoTypeId=\"Host\"");
    System.out.print("This ex. retrieves the attributes of: " +
dn);
    System.out.println("and prints them on the screen.");
    String className = new String("topoType");

    // instantiate a MOHandle with the object name and class
    // call an MGet to retrieve all the information about the
MOHandle
    // from the mis into the MOHandle.

    moh = new MOHandle(dn, className, plat);
    moh.MGet(TIMEOUT);
```

This code segment creates the `MOHandle` object `moh`, and uses its `MGet` method to retrieve all the attribute values of the MOIs it represents from the network.

For more information about the `MOHandle` class, refer to Chapter 2 “Java PMI API” in *Java API Reference*.

2.3.2 Instantiating `MOHCollectionByRule` and `MOHCollectionEnum` Classes

When creating JMI applications, it is often more efficient to treat a set of MOIs as one collection. In addition, it is sometimes necessary to maintain information about a set of MOIs in an MIS and be able to manage the set as a whole.

For example, your application may contain a GUI element that displays the MOIs in a particular subtree of the management information tree (MIT) in an MIS. The GUI element will have to refresh the screen whenever a change occurs in the subtree it represents. In addition, your application may allow network operators to delete or change the attributes of a particular subtree in the MIT. See the following figure.

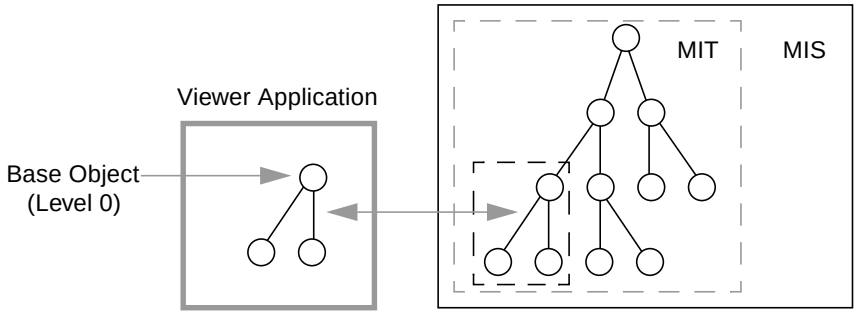

FIGURE 2-5 Example of `MOHCollectionByRule` Usage

Although you can use `MOHandle` objects to perform these operations, the burden is on you to provide the necessary mechanism for creating and maintaining sets of these objects.

For this reason, the JMI API provides the `MOHCollectionByRule` and the `MOHCollectionEnum` classes. These are similar classes that allow you to represent a group of `MOHandle` objects. The following table compares the two classes.

TABLE 2-1 `MOHCollectionByRule` versus `MOHCollectionEnum`

Class	Description
`MOHCollectionByRule`	• Member `MOHandle` objects satisfy a particular rule • Member `MOHandle` objects cannot be user-manipulated • Tracks all the `MOHandles` within the collection, based on changes in the network for these `MOHandles`
`MOHCollectionEnum`	• Member `MOHandle` objects are added with no constraints • Member `MOHandle` objects can be user-manipulated

▼ To Instantiate the MOHCollectionByRule Class

- **Provide the following parameters:**
  - Base object: Specifies the root object of the collection.
  - Scope: Defines the scope of the MIT based on the Base object.
    - A value of LV(*n*), where *n* is an Integer, specifies the nth level of the tree whose root is specified by the base object parameter. LV(0) is the same as the Base object.
    - A value of TO(*n*), where *n* is an Integer, specifies a tree of *n* levels whose root is the Base object.
  - Filter: Specifies the criteria to use for passing events and automatically updates the collection. For example, you can define a filter so that any event taking place in the first four levels (0 to 3) of a collection will be passed to the collection's event listener. If any MOI that the collection monitors gets deleted, the collection will be updated accordingly to maintain an up-to-date view of the collection.
  - Platform: Specifies the Platform object that allows you to gain access to the MIS.

Following is an example of a code segment that creates an MOHCollectionByRule object.

CODE EXAMPLE 2-2 Creating an MOHCollectionByRule Object

```
MOHCollectionByRule app_instances = null;
      String base = new String("subsystemId=\"EM-MIS\"");
      System.out.println("Base object is " + base);
      String scope = new String("LV(1)");
      System.out.println("Scope is " + scope);
      String filter = new String("CMISFilter(item:equality:
           {objectClass, emApplicationInstance})");

      System.out.println("Filter used " + filter);
      // get all app instance objects
      try {
        app_instances =
            new MOHCollectionByRule(base, scope, filter, plat);
      System.out.println("constructed album");
        app_instances.populate(TIMEOUT);
      System.out.println("populated the MOHCollectionByRule ");
```

This code segment creates the MOHCollectionByRule object app_instances using the base, scope, filter, and plat objects as parameters and populates it using the populate method.

▼ To Instantiate the MOHCollectionEnum Class

- **Provide one or both of the following parameters:**
  - MOHCollectionByRule object: This object is used to initialize the new MOHCollectionEnum object with the member MOHandle objects that MOHCollectionByRule contains.
  - Platform: The Platform object that allows you to gain access to the MIS.

Note – If only the platform parameter is specified, the new MOHCollectionEnum collection will not contain any MOHandle object.

2.4 Registering Event Listeners

The most important aspect of a JMI application is the ability to be able to update information about the MOIs the application is managing. This is done by registering event listeners and defining the handlers (discussed in Section 2.5 “Handling Events” on page 2-13) that will be invoked when an event indicating a change in the network occurs.

The JMI API allows you to register six types of events as shown in the following table:

TABLE 2-2 Event Types

Event	Description
AttributeValueChange	Gets generated when one or more attribute values change for an MOI
ObjectCreation	Gets generated when an MOI object is created
ObjectDeletion	Gets generated when an MOI object is deleted
RawEvent	Any event
MOHandleIncluded	Gets generated when an object is added to a collection
MOHandleExcluded	Gets generated when an object is deleted from a collection

These events can be registered against `Platform`, `MOHandle`, `MOHCollectionByRule`, and `MOHCollectionEnum` objects as shown in the following table.

TABLE 2-3 Event Type Mapping

Event Level	Description
`Platform`	You can register Callbacks for the following events that affect any object in the network defined by a `Platform` object: • Attribute value change events • Object deletion events • Object creation events • Raw events
`MOHandle`	Registers Callbacks for the following events that affect the MOI represented by an `MOHandle` object: • Attribute value change events • Object deletion events • Object creation events • Raw events
`MOHCollectionByRule`	Registers Callbacks for the following events that affect any `MOHandle` object in an `MOHCollectionByRule` object's collection: • Attribute value change events • Object deletion events • Object creation events • Raw events • `MOHandle` included events • `MOHandle` excluded events
`MOHCollectionEnum`	Registers Callbacks for the following events that affect any `MOHandle` object in an `MOHCollectionEnum` object's collection: • Attribute value change events • Object deletion events • Object creation events • Raw events

▼ To Register an Event Listener

1. **Define a class that implements the `EventReportListener` interface of the JMI API.**

   Part of defining this class is to define the interface's `handler` method that will be invoked when the corresponding event arrives (see Section 2.5 "Handling Events" on page 2-13).

2. **Register for Callbacks for the events that you're interested in using the appropriate methods of the `Platform`, `MOHandle`, `MOHCollectionByRule`, or `MOHCollectionEnum` objects.**

   For example:

   - To register an `AttributeValueChange` event listener for a `Platform` object, use its `addAttributeValueChangeListener` method.
   - To register an `ObjectCreation` event listener for an `MOHandle` object, use its `addObjectCreationListener` method.
   - To register an `MOHandleIncluded` event listener for an `MOHCollectionByRule` object, use its `addMOHandleIncludedListener` method.

   Following is an example of a code segment that registers `AttributeValueChange`, `ObjectCreation`, and `ObjectDeletion` event listeners.

CODE EXAMPLE 2-3 Registering Event Listeners

```
public class MOHandleEvent implements EventReportListener {

    public static final double TIMEOUT = 3600.0;
    static Platform plat = null;
..................
..................
..................
// register for Callbacks for AVC, ObjectCreation and
// ObjectDeletion
System.out.println("MOHandle is logId=\"AlarmLog\"");
System.out.println("Registering for events for MOHandle  ");
moh.addAttributeValueChangeListener(gt);
moh.addObjectCreationListener(gt);
moh.addObjectDeletionListener(gt);
System.out.println("Waiting for events ...");

// wait here as the events arrive from mis.
// user can modify an attribute here for AlarmLog to
// verify the event delivery.
..................
..................
..................
```

For more information about registering event listeners, refer to the method descriptions of the `Platform`, `MOHandle`, `MOHCollectionByRule`, and `MOHCollectionEnum` classes in Chapter 2 “Java PMI API” in *Java API Reference*.

2.5 Handling Events

In addition to registering event listeners, you need to define the handlers that will be invoked in response to an event. For each listener object you create, you must define the body of its `handler` method.

CODE EXAMPLE 2-4 defines an event listener’s `handler` method.

The following code prints the event type, the name of the object that generated the event, the name of the class of the object that generated the event, and, in the case of a raw event, information about the event.

CODE EXAMPLE 2-4 Defining a `handler`

```
    public void handler(EventReport ind) {
        String type = ind.getName();
        System.out.println("Event Type = " + type);
        System.out.println("Object Name = " + ind.getMOName());
        System.out.println("Object Class = " + ind.getMOClass());
                try {
                    System.out.println("Event Info = " +
ind.getInfo());
                    AbstractData abs = ind.getInfoRaw();
                    System.out.println("Event Info Raw = " +
abs.getStr());
             } catch (JmiException ex) { ex.printStackTrace(); }
    }
```

2.6 C++ Equivalents for the JMI API Classes

The following table shows the C++ PMI classes that are equivalent to the JMI API classes.

TABLE 2-4 C++ Equivalents for JMI API Classes

JMI API Class	Equivalent C++ PMI Class
`Platform`	`Platform`
`MOHandle`	`Image`
`MOHCollectionByRule`	`Album`
`MOHCollectionEnum`	`Album`
`AbstractData`	`Morf`
`EventReport`	`CurrentEvent`

2.7 Sample JMI Application

This section lists the code of the following five sample applications that use the JMI API:

- `PlatformEvent.java` (CODE EXAMPLE 2-5)
- `MOHandleTest.java` (CODE EXAMPLE 2-6)
- `MOHandleEvent.java` (CODE EXAMPLE 2-7)
- `EmWho.java` (CODE EXAMPLE 2-8)
- `CollectionEvent.java` (CODE EXAMPLE 2-9)

2.7.1 PlatformEvent.java

The following code example shows you how to:

- Instantiate the `Platform` class.
- Register Callbacks for attribute value change, object creation, and object deletion.
- Verify event delivery.

CODE EXAMPLE 2-5 PlatformEvent.java

```
/*
 * Copyright (c) 12/08/97 Sun Microsystems, Inc.
 * All Rights Reserved.
 */

import com.sun.em.api.pmi.*;
import java.net.*;
import java.util.*;

public class PlatformEvent implements EventReportListener {

    public static final double TIMEOUT = 3600.0;

    static void usage() {
    System.err.println("Usage:");
    System.err.println("java PlatformEvent <servername> <mis-
name> <user-name> <password>");
    System.err.println("\t-Run the example with <servername> as
the remote server and <misname> where EM mis is running.");
    System.exit(-1);
    }

    public static void main(String args[]) {

    if (args.length < 4)
        usage();

    PlatformEvent gt0, gt1, gt2;
    MOHandle moh = null;

    try {
        gt0 = new PlatformEvent(args[0], args[1], args[2],
args[3]);
     }
    catch (JmiException ex) {System.out.println(ex); }
    }

    public void handler(EventReport ind) {
```

CODE EXAMPLE 2-5 PlatformEvent.java *(Continued)*

```
        String type = ind.getName();
        System.out.println("Event Type = " + type);
        System.out.println("Object Name = " + ind.getMOName());
        System.out.println("Object Class = " + ind.getMOClass());
                try {
                    System.out.println("Event Info = " +
ind.getInfo());
                    AbstractData abs = ind.getInfoRaw();
                    System.out.println("Event Info Raw = " +
abs.getStr());
              } catch (JmiException ex) { ex.printStackTrace(); }
    }


    public PlatformEvent(String host, String mis, String user,
String passwd) throws JmiException {

    try {
    // instantiate Platform
    plat = new Platform(host, mis, user, passwd);
    // register for events
    System.out.println("Registering for events for Platform
...");
    plat.addAttributeValueChangeListener(this);
    plat.addObjectCreationListener(this);
    plat.addObjectDeletionListener(this);
    System.out.println("Callbacks registered");
    System.out.println("Waiting for events ...");

    }
    catch (JmiException ex) { throw ex; }

    }

    Platform plat = null;
    private static final String sccsID =
        "@(#)PlatformEvent.java1.4 97/12/08 Sun Microsystems,
Inc.";
}
```

2.7.2 MOHandleTest.java

The following code example shows you how to:

- Get attributes of an `MOHandle`.
- Create an `MOHandle` (`logId="testLog"`) after setting attributes.

CODE EXAMPLE 2-6 MOHandleTest.java

```
/*
 * Copyright (c) 01/27/98 Sun Microsystems, Inc. All Rights
Reserved.
 */
import com.sun.em.api.pmi.*;
import java.net.*;
import java.util.*;

public class MOHandleTest {

    public static final double TIMEOUT = 3600.0;
    static Platform plat = null;

    static void usage() {
    System.err.println("Usage:");
    System.err.println("java MOHandleTest <servername> <mis-name>
<username> <password>");
    System.err.println("\t-Run the example with <servername> as
the remote server and <misname> where EM mis is running.");
    System.exit(-1);
    }

    public static void main(String args[]) {

    MOHandleTest gt = null;
    MOHandle moh = null;
    if (args.length < 4) usage();
    try { gt = new MOHandleTest();

    // instantiate Platform with the supplied server name and
    // mis name.

    plat = new Platform(args[0],args[1], args[2],args[3]);

        String [] attrNames = null;
    String dn = new String("topoTypeDBId=NULL/
topoTypeId=\"Host\"");
    System.out.print("This example retrieves the attributes of: "
+ dn);
```

CODE EXAMPLE 2-6 MOHandleTest.java *(Continued)*

```
    System.out.println("and prints them on the screen.");
    String className = new String("topoType");

    // instantiate a MOHandle with the object name and class
    // call a MGet to retrieve all the information about the
MOHandle
    // from the mis into the MOHandle.

    moh = new MOHandle(dn, className, plat);
    moh.MGet(TIMEOUT);

    // verify that the object exists in the network.

    if (moh.exists() == false)
        System.out.println(dn + "does not exist");

    // Get the attribute names of the MOHandle
         attrNames = moh.getAttrNames();


    System.out.println("The attributes for the object are ..");

         for (int i=0; i <attrNames.length; i++)
             System.out.println("Attribute: " + attrNames[i]);


    // second part of the example. Create a object in the mis.
    String testdn = new String("logId=\"testLog\"");
    String testclass = new String("log");
    System.out.println("Create a object " + testdn + "in mis");
    // instantiate the MOHandle and retrieve the attribute info.
    MOHandle mo = new MOHandle(testdn, testclass, plat);
    mo.MGet(TIMEOUT);

    if (mo.exists()) {
        System.out.println("object exists! quitting ...");
        System.exit(11);
    }

    mo.setStr("logFullAction", "wrap");
    System.out.println("set logFullAction");

    mo.setStr("administrativeState", "unlocked");
    System.out.println("set administrativeState");

    mo.setStr("maxLogSize", "1000000");
    System.out.println("set maxLogSize");
```

CODE EXAMPLE 2-6 MOHandleTest.java *(Continued)*

```
    mo.setStr("discriminatorConstruct", "or: {}");
    System.out.println("set discriminatorConstruct");

    mo.MCreate(TIMEOUT);
    }
    catch (JmiException ex) {System.out.println(ex); }
    System.exit(2);
    }


    public MOHandleTest() throws JmiException { }
   private static final String sccsID =
       "@(#)MOHandleTest.java1.4 98/01/27 Sun Microsystems,
Inc.";

}
```

2.7.3 MOHandleEvent.java

The following code example shows you how to:

- Register Callbacks for an AttributeValueChange, ObjectCreation, and ObjectDeletion events or an MOHandle object.
- Verify event delivery.

CODE EXAMPLE 2-7 MOHandleEvent.java

```
/*
 *  Copyright (c) 05/05/98 Sun Microsystems, Inc. All Rights
Reserved.
 */

import com.sun.em.api.pmi.*;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.net.*;
import java.util.*;

public class MOHandleEvent implements EventReportListener {

    public static final double TIMEOUT = 3600.0;
    static Platform plat = null;

    static void usage() {
```

CODE EXAMPLE 2-7 MOHandleEvent.java *(Continued)*

```
    System.err.println("Usage:");
    System.err.println("java MOHandleEvent <servername> <mis-
name> <username> <password>");
    System.err.println("\t-Run the example with <servername> as
the remote server and <misname> where EM mis is running.");
    System.exit(-1);
    }

    public static void main(String args[]) {

    if (args.length < 4)
        usage();

    MOHandleEvent gt = null;
    MOHandle moh = null;
    String dn = new String("logId=\"AlarmLog\"");
    String className = new String("log");


    try { gt = new MOHandleEvent();
        // instantiate the platform.

        plat = new Platform(args[0], args[1], args[2], args[3]);

        // nstantiate the MOHandle and retrieve the attribute info
        moh = new MOHandle(dn, className, plat);
        moh.MGet(TIMEOUT);

        // register for Callbacks for AVC, ObjectCreation and
        // ObjectDeletion
        System.out.println("MOHandle is logId=\"AlarmLog\"");
        System.out.println("Registering for events for MOHandle
");
        moh.addAttributeValueChangeListener(gt);
        moh.addObjectCreationListener(gt);
        moh.addObjectDeletionListener(gt);
        System.out.println("Waiting for events ...");

        // wait here as the events arrive from mis.
        // user can modify an attribute here for AlarmLog to
        // verify the event delivery.
    }
    catch (JmiException ex) {System.out.println(ex); }
    catch (Exception ex) {System.out.println(ex); }

    }
```

CODE EXAMPLE 2-7 MOHandleEvent.java *(Continued)*

```
    public void handler(EventReport ind) {
    String type = ind.getName();
    System.out.println("Event Type = " + type);
    System.out.println("Object Name = " + ind.getMOName());
    System.out.println("Object Class = " + ind.getMOClass());
    }


    public MOHandleEvent() throws JmiException { }

   private static final String sccsID =
       "@(#)MOHandleEvent.java1.7 98/05/05 Sun Microsystems,
Inc.";

}
```

2.7.4 EmWho.java

The following code example shows you how to:

- Create a collection and populate it.
- Print out the members of the collection, which are the application instances connected to MIS.

CODE EXAMPLE 2-8 EmWho.java

```
/*
 * Copyright (c) 05/05/98 Sun Microsystems, Inc. All Rights
Reserved.
 */
import com.sun.em.api.pmi.*;
import java.net.*;
import java.util.*;

public class EmWho {

    public static final double TIMEOUT = 3600.0;
    static Platform plat = null;

    static void usage() {
    System.err.println("Usage:");
    System.err.println("java EmWho <servername> <mis-name>
        <username> <password>");
    System.err.println("\t-Run the example with <servername> as
the remote server and <misname> where EM mis is running.");
```

CODE EXAMPLE 2-8 EmWho.java *(Continued)*

```
    System.exit(-1);
    }

    public static void main(String args[]) {
      EmWho ew = null;
      if (args.length < 4)
    usage();
      try {
        ew = new EmWho();
    System.out.println("Instantiating platform");
        plat = new Platform(args[0],args[1] , args[2],args[3]);

        ew.get_users();
      }
      catch (JmiException ex) {System.out.println(ex); }

    }

        public void get_users() {

      System.out.println("Findout the applications connected to
mis");
      MOHCollectionByRule app_instances = null;
      String base = new String("subsystemId=\"EM-MIS\"");
      System.out.println("Base object is " + base);
      String scope = new String("LV(1)");
      System.out.println("Scope is " + scope);
      String filter = new String("CMISFilter(item:equality:
{objectClass, emApplicationInstance})");

      System.out.println("Filter used " + filter);
      // get all app instance objects
      try {
        app_instances = new MOHCollectionByRule(base, scope,
filter, plat);
      System.out.println("constructed album");
        app_instances.populate(TIMEOUT);
      System.out.println("populated the MOHCollectionByRule ");

        MOHandle[] mohs = null;
        mohs = app_instances.getMOHandles();
        System.out.println("# of app instance objects = " +
mohs.length);

        //
        // snarf user name out of emUserID attribute
        //
```

CODE EXAMPLE 2-8 EmWho.java *(Continued)*

```
        for (int i=0; i < mohs.length; i++) {
            mohs[i].MGet(TIMEOUT);
            System.out.println(mohs[i].getObjectName());

System.out.println(mohs[i].getStr("emApplicationType"));
        }


      }
      catch (JmiException ex) {
        System.out.println(ex);
        System.exit(1);
      }

         }
   private static final String sccsID =
        "@(#)EmWho.java1.5 98/05/05 Sun Microsystems, Inc.";

}
```

2.7.5 CollectionEvent.java

The following code example shows you how to:

- Register Callbacks for AttributeValueChange, ObjectCreation, and ObjectDeletion events for a collection.
- Verify event delivery.

CODE EXAMPLE 2-9 CollectionEvent.java

```
/*
 * Copyright (c) 05/05/98 Sun Microsystems, Inc. All Rights
Reserved.
*/

import com.sun.em.api.pmi.*;
import java.net.*;
import java.util.*;

public class CollectionEvent implements EventReportListener {

    public static final double TIMEOUT = 3600.0;
    static Platform plat = null;
```

CODE EXAMPLE 2-9 CollectionEvent.java *(Continued)*

```
    static void usage() {
    System.err.println("Usage:");
    System.err.println("java CollectionEvent <servername> <mis-
name> <username> <password>");
    System.err.println("\t-Run the example with <servername> as
the remote server and <misname> where EM mis is running.");
    System.exit(-1);
    }

    public static void main(String args[]) {

    if (args.length < 4)
        usage();
    CollectionEvent gt = null;
    MOHCollectionByRule mcl = null;
    String base = new String("/systemId=name:\""+ args[1] + "\"");
//  String base = null;
    String scope = new String("LV(1)");
    String filter = new String("");

    System.out.println("Populate the MOHandles for: ");
    System.out.println(base);

    try { gt = new CollectionEvent();

    plat = new Platform(args[0], args[1], args[2], args[3]);;
    System.out.println("Platform instantiated");
    }
    catch (JmiException ex) { System.out.println(ex); }

    try { mcl = new MOHCollectionByRule(base, scope, filter, plat);
}
    catch (JmiException ex) {
        System.out.println(ex);
        System.exit (1);
    }

    System.out.println("Collection created");

    try { mcl.populate(TIMEOUT); }
    catch (JmiException ex) { System.out.println(ex); }
    System.out.println("Collection populated");

    try {
    System.out.println("Scope: " + mcl.getScope());
    System.out.println("Filter: " + mcl.getFilter());
```

CODE EXAMPLE 2-9 CollectionEvent.java *(Continued)*

```
    System.out.println("Base Object: " +
mcl.getBaseManagedObject());
    }
    catch (JmiException ex) {}

    MOHandle[] mohs = null;
        try { mohs = mcl.getMOHandles(); }
        catch (JmiException ex) { System.out.println(ex); }

    System.out.println("The # of MOHandles are .." + mohs.length);
    System.out.println("The MOHandles for the Collection are ..");
    try {
        for (int i=0; i <mohs.length; i++) {
        System.out.println("Object Name: " +
mohs[i].getObjectName()); }
    }
    catch (JmiException ex) {System.out.println(ex); }

        System.out.println("Got all the MOHandles");

        try {

    mcl.MGet(TIMEOUT);
    // set the collection in tracking mode
    mcl.setTracking(true);

    // register for events
    // MOHandleIncluded and MOHandleExcluded
        mcl.addMOHandleIncludedListener(gt);
        mcl.addMOHandleExcludedListener(gt);
    System.out.println("Wait for events ..");
    }
        catch (JmiException ex) {System.out.println(ex); }

    }


    public CollectionEvent() throws JmiException {

    }

    public void handler(EventReport ind) {
        String type = ind.getName();
        System.out.println("Event Type = " + type);
        System.out.println("Object Name = " + ind.getMOName());
        System.out.println("Object Class = " + ind.getMOClass());
```

CODE EXAMPLE 2-9 `CollectionEvent.java` *(Continued)*

```
        try {
            System.out.println("Object Raw info = " +
                               ind.getInfoRaw().getStr());
        } catch (JmiException ex) { ex.printStackTrace(); }
    }

  private static final String sccsID =
       "@(#)CollectionEvent.java1.7 98/05/05 Sun Microsystems,
Inc.";

}
```

CHAPTER 3

Using the Java Alarm API

The Java Alarm API defines the classes and methods which allow you to create applications that perform alarm management operations. This chapter covers the most important aspects of using the API and provides detailed examples. For more information about the API, refer to *Java API Reference*.

The following topics are covered in this chapter:

These topics are arranged, as far as possible, to reflect the order you need to follow when creating alarm management applications. For example, before getting alarms, you must have associated an `AlarmLog` object with a log, created a query, and specified the attributes to be returned. This sequence becomes clear as you go through this chapter.

3.1 Overview

Alarms are events indicating abnormal conditions that the MIS logs (through the logging subsystem) into alarm logs (the default log object is `AlarmLog`). Each log entry (record) represents an alarm and is created based on the mapping provided in `evv2oclist` object in the MIS and the filter created for the log. An alarm:

- Can be logged.
- Has attributes that allows it to be acknowledged and cleared.
- Is recognized by and displayed in the Solstice Enterprise Manager (Solstice EM) Alarms window tool.

When the MIS receives an event, it uses a predefined filter to determine whether the event is an alarm and logs it accordingly as an alarm record. You can use the Alarms window to view and monitor alarms.

The Java Alarm API provides a programmatic interface to achieve functionality similar to the Alarms window. The Java Alarm API hides the implementation details of how logs stored in databases are accessed and processed, and provides the classes and methods that allow you to create alarm management applications. If the implementation details change, your application still works. See the following figure.

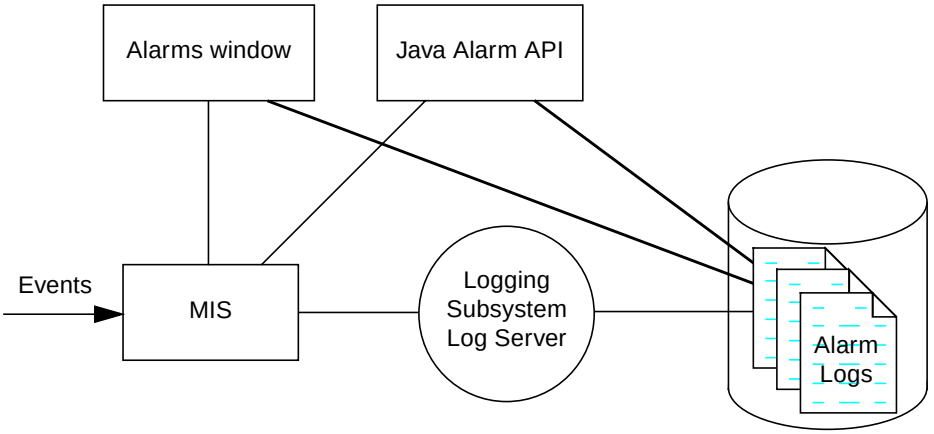

FIGURE 3-1 The Java Alarm API

Note – The Java Alarm API works only on alarms that are derived from the GDMO object `emAlarmRecord` and does *not* work with user-defined alarms.

3.1.1 Alarm Management Tasks

The Java Alarm API allows you to perform the following alarm management tasks:

- Monitor alarms
- Acknowledge alarms
- Take actions based on alarms
- Clear alarms

These tasks can be performed using the following main classes of the Java Alarm API:

- `AlarmLog` class
- Query classes (`FilterItem`, `Filter`, and `GenericQuery`)
- Listener classes (such as `AlarmLogListener` and `AlarmLogCreationListener`)

See the following figure.

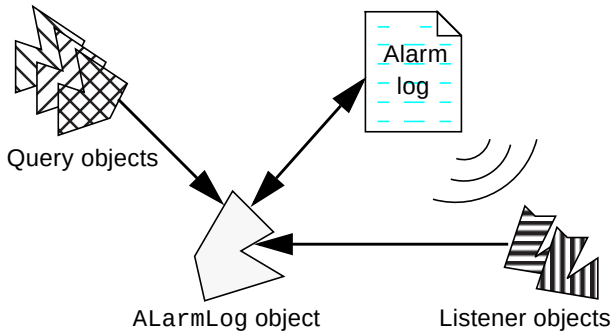

FIGURE 3-2 Main “Players” in the Java Alarm API

Each instance of the `AlarmLog` class represents an alarm log and contains methods for accessing and modifying alarm information. The query classes provide the filtering mechanism needed by some of the `AlarmLog` class methods. The listener classes provide the mechanism for reacting to alarms.

3.1.2 Alarm Management Task Flow

When creating alarm management applications, keep in mind the following model of task flow, as illustrated in the following figure.

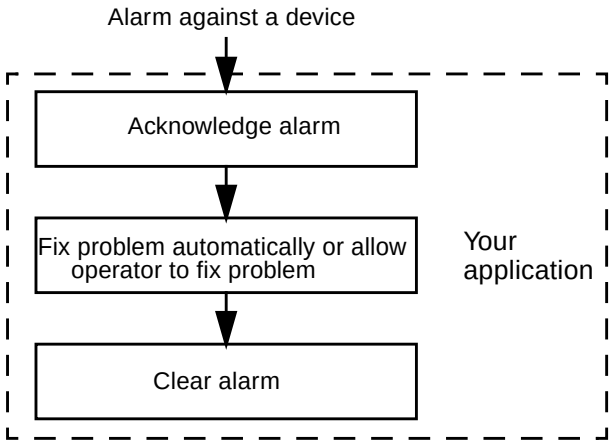

FIGURE 3-3 Flow of Tasks When Handling an Event

3.2 Instantiating the `AlarmLog` Class

The first step to take when performing alarm operations is to instantiate the `AlarmLog` class. This class represents the MIS's log objects that contain alarms. Each instance of this class represents one log. See the following figure.

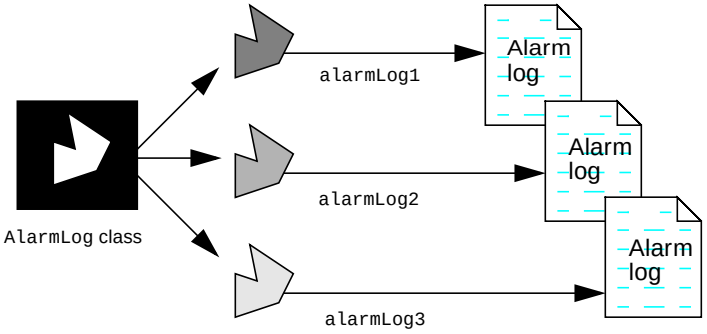

FIGURE 3-4 `AlarmLog` Objects

▼ To Instantiate the AlarmLog Class

1. **Create a** LogName **object as shown in the following example:**

```
LogName logName = new LogName(MISName, “AlarmLog”);
```

The logName object allows you to hide the naming internals of the log. When creating the logName object, you must supply the following two arguments:

- MISName: The name of the MIS on which the log exists.
- LogName: The name of the log (in this case AlarmLog).

Note – Don’t confuse “AlarmLog”, which is the name of the default log object in Solstice EM, with the AlarmLog class.

---

2. **Create an** AlarmLog **object as shown in the following example:**

```
AlarmLog log = new AlarmLog(platform, logName);
```

The AlarmLog object constructor takes the following parameters:

- platform: An instance of the Platform class that represents the MIS to which the application is connected.
- logName: A LogName object representing a log’s name.

Note – The MIS name parameter used to create the LogName object and the MIS which the platform parameter represents when creating the AlarmLog object can be different.

3.3 Creating Query Objects

The Java Alarm API allows you to create filter objects that you can use to query alarm logs. For example, you can create queries to retrieve all alarm records that have a perceived severity that is major or critical.

▼ To Create a Query Object

1. **Create one or more `FilterItem` objects that define the conditions to check for in the query.**

   This topic is described in Section 3.3.1 “Creating FilterItem Objects” on page 3-7.

2. **Create a Filter object that combines the `FilterItem` objects you defined in Step 1 based on a logical criteria (`AND` or `OR`).**

   This topic is described in Section 3.3.2 “Creating Filter Objects” on page 3-8.

3. **Create a `GenericQuery` object that combines the `Filter` objects you defined in Step 2 based on a logical criteria (`AND` or `OR`). See the following figure.**

   This topic is described in Section 3.3.3 “Creating GenericQuery Objects” on page 3-9.

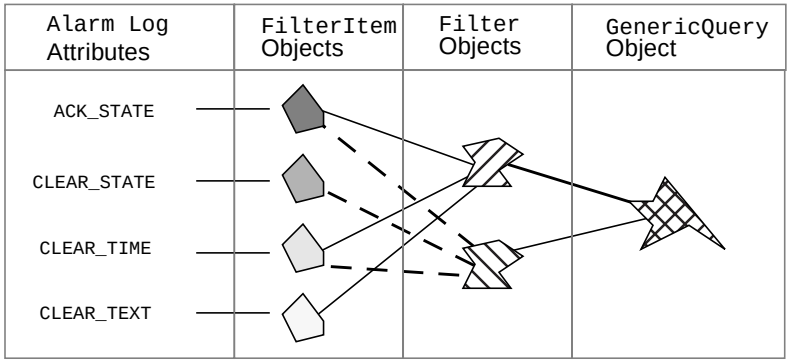

FIGURE 3-5 Query Objects

As shown in FIGURE 3-5, the `GenericQuery` object is used to join various `Filter` objects and the `Filter` object is used to join various `FilterItem` objects. Constructing a `GenericQuery` object in this fashion hides the implementation details of filtering, allows `FilterItem` and `Filter` objects to be reused in the same application to create different queries, and allows you to create complicated queries.

3.3.1 Creating `FilterItem` Objects

A `FilterItem` object represents a condition in a query that affects only one alarm record attribute. For example, you might want to retrieve all alarm records whose `PERCEIVED_SEVERITY` attribute is equal to `EMSeverity.MAJOR` or `EMSeverity.CRITICAL`.

To create a `FilterItem` object, you need to provide the following three parameters:

- Attribute name: Name of the alarm record attribute to be checked by the condition. See Section 3.4 “Creating AlarmRecordAttributeSet Objects” on page 3-10 for more information on alarm record attributes.
- Relational operator: A `RelationCriteria` object that defines a relation operator for the condition (=, >,<).
- Attribute values: One or more attribute values.

Following is an example of a code segment that creates a `FilterItem` object:

```
EMSeverity[] sev = {
                EMSeverity.MAJOR,
                EMSeverity.CRITICAL
            };
FilterItem filterItem1 =
new FilterItem(AlarmRecordAttribute.PERCEIVED_SEVERITY,
RelationCriteria.EQUAL, sev);
```

This code creates a `FilterItem` object based on the severity of the alarm record.

The first statement in the code above defines the attribute values that will be used when creating the `FilterItem` object `filterItem1`.

The second statement creates the `FilterItem` object `filterItem1`. All alarm records whose perceived severity is major or critical will pass this filter.

3.3.2 Creating `Filter` Objects

A `Filter` object implements an alarm filter. You construct a filter by combining an instance of the `LogicalCriteria` object with a list of `FilterItem` objects. In other words, an alarm filter consists of one or more conditions combined together by the logical operators `AND` or `OR`.

To create a `Filter` object, you need to provide the following two parameters:

- `LogicalCriteria` object: The logical operator (`AND` or `OR`) to be used in the filter.
- `FilterItem` object: The object specifying a condition to check for. You can add more `FilterItem` objects using the `addFilter` method of the `Filter` object.

Note – If you don't specify a logical criteria, `AND` is assumed by default.

Following is a code segment that creates a `Filter` object:

```
LogicalCriteria logCriteria1 = new LogicalCriteria OR;
Filter filter = new Filter(logCriteria1, filterItem1);
```

In this code segment, the first statement defines `OR` as the logical operator to be used when creating the filter.

The second statement creates the `Filter` object filter based on the logical operator defined in the previous statement and the `FilterItem` object defined in Section 3.3.1 "Creating FilterItem Objects" on page 3-7.

To add more `FilterItem` objects to the filter object, you can use the `addFilter` method as follows:

```
filter.addFilterItem(filterItem2);
filter.addFilterItem(filterItem3);
```

In this case, the filter object consists of three conditions combined by the `OR` operator (`filterItem1 OR filterItem2 OR filterItem3`).

3.3.3 Creating `GenericQuery` Objects

A `GenericQuery` object implements a generic alarm filter. You construct a `GenericQuery` object by combining a `LogicalCriteria` object with `FilterItem` objects.

To create a `GenericQuery` object, you need to provide the following two parameters:

- `LogicalCriteria` object: The logical operator (`AND` or `OR`) to be used in the filter.
- `Filter` object: The object specifying one or more conditions to be checked. You can add more `Filter` objects using the `addFilter` method of the `GenericQuery` object.

Note – If you don't specify a `LogicalCriteria` object, `AND` is assumed by default.

Following is a code segment that creates a `GenericQuery` object:

```
GenericQuery query = new GenericQuery();
query.addFilter(filter1);
query.addFilter(filter2);
```

In this code segment, the first statement creates a new `GenericQuery` object (`query`). The second and third statements add two `Filter` objects to the query. These two filters are combined using the `AND` operator.

3.4 Creating AlarmRecordAttributeSet Objects

Each record in an alarm log consists of 22 fields or attributes. However, in many instances you only need to retrieve or modify a subset of these attributes, which you can specify using the AlarmRecordAttributeSet objects. These objects are used by the API methods that retrieve or modify alarm record attributes.

The following table contains a list of the 22 alarm record attributes that are represented as variables in the AlarmRecordAttribute class:

TABLE 3-1 AlarmRecord Attributes

Attribute	Description
ACK_OPERATOR	User ID of the operator who acknowledged the alarm
ACK_STATE	Specifies whether the alarm is acknowledged
ACK_TEXT	Text added when the alarm was acknowledged
ACK_TIME	Time when the alarm was acknowledged
CLEAR_OPERATOR	User ID of the operator who cleared the alarm
CLEAR_STATE	Specifies whether the alarm is cleared
CLEAR_TEXT	Text added when the alarm was cleared
CLEAR_TIME	Time when the alarm was cleared
DISPLAY_OPERATOR	User ID of the operator who made a change to the display state flag
DISPLAY_STATE	A flag that can be used by applications to determine whether to display an alarm
DISPLAY_TEXT	Text added when the alarm display flag is changed
DISPLAY_TIME	Time when the alarm display state was changed
EVENT_TIME	Time when the event occurred
EVENT_TYPE	Type of event
LOGGING_TIME	Time when the event got logged
LOG_RECORD_ID	ID of the log record created
MANAGED_OBJECT_INSTANCE	Actual object that raised this alarm
PERCEIVED_SEVERITY	Severity of the alarm
PROBABLE_CAUSE	Probable cause of the alarm

TABLE 3-1 `AlarmRecord` Attributes *(Continued)*

Attribute	Description
`ADDITIONAL_TEXT`	Additional information about the alarm
`LOG_NAME`	Name of the log in which this alarm is logged
`MIS_NAME`	MIS name on which this log exists

To define a set of alarm record attributes, create an `AlarmRecordAttributeSet` object and add the required attributes (using the `add` method) as shown in the following code example:

CODE EXAMPLE 3-1 Defining Alarm Record Attributes

```
AlarmRecordAttributeSet attrSet = new AlarmRecordAttributeSet();
attrSet.add(AlarmRecordAttribute.LOG_RECORD_ID);
attrSet.add(AlarmRecordAttribute.PERCEIVED_SEVERITY);
attrSet.add(AlarmRecordAttribute.PROBABLE_CAUSE);
attrSet.add(AlarmRecordAttribute.CLEAR_STATE);
attrSet.add(AlarmRecordAttribute.CLEAR_TIME);
attrSet.add(AlarmRecordAttribute.CLEAR_TEXT);
attrSet.add(AlarmRecordAttribute.CLEAR_OPERATOR);
attrSet.add(AlarmRecordAttribute.ACK_STATE);
attrSet.add(AlarmRecordAttribute.ACK_TIME);
attrSet.add(AlarmRecordAttribute.ACK_TEXT);
attrSet.add(AlarmRecordAttribute.LOGGING_TIME);
attrSet.add(AlarmRecordAttribute.EVENT_TYPE);
attrSet.add(AlarmRecordAttribute.MANAGED_OBJECT_INSTANCE);
attrSet.add(AlarmRecordAttribute.LOG_NAME);
attrSet.add(AlarmRecordAttribute.ADDITIONAL_TEXT);
attrSet.add(AlarmRecordAttribute.DISPLAY_OPERATOR);
attrSet.add(AlarmRecordAttribute.DISPLAY_STATE);
attrSet.add(AlarmRecordAttribute.DISPLAY_TIME);
attrSet.add(AlarmRecordAttribute.EVENT_TIME);
```

3.5 Getting Alarms

Once the `AlarmLog` and query objects are created, you can perform alarm management operations. This section describes how to get:

- Alarm counts
- Alarms
- Alarms in batches

3.5.1 Getting Alarm Counts

To get the number of alarms in an alarm log that satisfy a particular query, use the `getAlarms` method of the `AlarmLog` class as shown in the following code segment:

```
count = log.getAlarmCount(query);
```

This code segment gets the alarm count based on `query`. If `query` is `null`, the `getAlarmCount` method returns the total number of alarms in the alarm log.

3.5.2 Getting Alarms

To retrieve the attributes of alarms that satisfy a particular query, use the `getAlarms` method of the `AlarmLog` object as shown in the following code segment:

```
AlarmRecord[] alarms = log.getAlarms(query, attrSet);
```

This code segment retrieves the alarm records that satisfy the query and stores them into the `alarms` array. The returned attributes contain the attributes specified by `attrSet` only.

3.5.3 Getting Alarms in Batches

It is often more efficient to retrieve alarm information in batches. This method frees the system to perform other operations while alarm information is being gathered.

To retrieve alarms in batches, you need to define a `BatchListener` object and pass it as an argument to the `getAlarmsInBatches` method. This method takes the following parameters:

- Query object: See Section 3.3.3 “Creating GenericQuery Objects” on page 3-9.
- Size: Batch size.
- `BatchListener` object: Implements methods (that you define) that tell the application what to do when alarm batches are received and when alarm batching is done.
- `AlarmRecordAttributeSet` object: See Section 3.4 “Creating AlarmRecordAttributeSet Objects” on page 3-10.

The following code example shows how to get alarms in batches.

CODE EXAMPLE 3-2 Getting Alarms in Batches

```
BatchListener batchList = new BatchListener("Critical");
System.out.println("Getting critical alarms in batches");
int batchId = log.getAlarmsInBatches(query, 5000, batchList,
attrSet);
System.out.println("Batch Id: " + batchId);
..........
..........
..........
class BatchListener implements AlarmsBatchListener {
    String id = null;
    public BatchListener(String id) {
         this.id = id;
    }
    public void batchReceived(int callId, AlarmRecord[] record) {
         System.out.println("BatchId: " + callId);
         System.out.println( "Number of alarms received: " +
record.length);

    }
    public void batchDone(int callId) {
        System.out.println("BatchId :" + callId);
        System.out.println("Batch complete");
    }
}
```

Notice that this code segment also contains the implementation of the `batchReceived` and `batchDone` methods of the `BatchListener` class. These methods specify what to do when a batch is received and when batching is done.

3.6 Clearing and Acknowledging Alarms

Clearing and acknowledging alarms is a routine task that you perform when managing alarms. The `AlarmLog` class provides the `setClearAlarms` and `setAckAlarms` methods that you can use to clear and acknowledge alarms. The following code example shows how to acknowledge and clear alarms.

CODE EXAMPLE 3-3 Clearing and Acknowledging Alarms

```
        ..............
        AlarmRecord[] alarms = log.getAlarms(q1, attrSet);
        ..............
        AlarmRecordId alrmId =alarms[0].getLogRecordId();
        AlarmRecordId[] alrm = {alrmId};
        log.setAckAlarms(alrm, "meaningful text needed");
        log.setClearAlarms(alrm, "meaningful text needed");
        ............
```

In this code segment, the `setAckAlarms` and `setClearAlarms` methods acknowledge and clear the alarm whose log id is specified by the `alarms` array (in this case the array contains only one id) by modifying the `ACK_TEXT` and `CLEAR_TEXT` attributes, respectively, of the corresponding alarm record in the corresponding log.

You can also clear alarms by sending an event using the `sendClearAlarmsEvent` method. This method sends an event that clears all alarms that originated from the same source all at once without having to clear them one-by-one, as in the `setClearAlarms` method.

```
log.sendClearAlarmsEvent(alrm);
```

3.7 Listening for Alarm Log Events

One of the main tasks in alarm management is to define how your application reacts to alarm records being created, deleted, or modified. This task is done by creating listener objects that define how to react to such events. These objects will be notified of such events by the logging subsystem.

The Java Alarm API provides the following four interfaces for creating alarm event listener objects:

- `AlarmLogCreationListener`: Contains one method (`alarmLogCreated`). You can define the body of this method to perform the appropriate actions when an alarm log record is *created*.
- `AlarmLogDeletionListener`: Contains one method (`alarmLogDeleted`). You can define the body of this method to perform the appropriate actions when an alarm log record is *deleted*.
- `AlarmLogModificationListener`: Contains one method (`alarmLogModified`). You can define the body of this method to perform the appropriate actions when an alarm log record is *modified*.
- `AlarmLogListener`: Contains three methods (`alarmLogCreated`, `alarmLogDeleted`, and `alarmLogModified`). You can define the body of these methods to perform the appropriate actions when an alarm log record is *created*, *deleted*, or *modified*.

▼ To Create an Alarm Event Listener

1. **Use the** `setEventAttrSet` **method to specify the list of attributes to be returned.**
2. **Use the** `addAlarmLogListener` **method to register to receive an event when the alarm log is created, deleted, or modified, as shown in the following example.**

```
//Listen for objectCreationEvent
//------------------------------
log.setEventAttrSet(attrSet);
System.out.println("Listening for alarm events");
log.addAlarmLogListener(new EventListener());
```

Note – If attributes other than the one specified in `attrSet` changes, an empty event is generated.

---

3. **Fill the bodies of the** `AlarmLogListener` **methods with the appropriate code (see** CODE EXAMPLE 3-5 **on page 3-20).**

3.8 Sample Programs

This section provides sample programs that perform alarm management.

3.8.1 AlarmBatch

This program gets alarms in batches. The query is based on `perceivedSeverity`, and the `batchSize` is `5000`. See the following code example.

CODE EXAMPLE 3-4 AlarmBatch

```
/*
 * Copyright 10/30/98 Sun Microsystems, Inc. All Rights Reserved.
 */
/*

  AlarmBatch <servername> <mis-name> <username> <password>
  <servername> - is the machine name on which the server is running.
  <mis-name> - is the machine name on which the mis is running.
  <username> - is the user login name.
  <password> - is the password of the user login.

*/
import com.sun.em.api.alarm.*;
import com.sun.em.api.common.*;
import java.util.Enumeration;
import java.util.Vector;
import com.sun.em.api.pmi.*;

public class AlarmBatch {

  static void usage() {
    System.err.println("Usage:");
    System.err.println("AlarmBatch  <servername> <mis-name>
<username> <password > ");
    System.exit(-1);
  }

  public static void main(String[] args) {
    try {
      new AlarmBatch(args);
    }
    catch (Exception e) {
```

CODE EXAMPLE 3-4 AlarmBatch *(Continued)*

```
      e.printStackTrace();
    }

    System.out.println("Done.");
    System.exit(0);

  }

  public AlarmBatch(String[] args)
    throws AlarmException {

      if (args.length < 4)
        usage();

     //Create an attribute set: The set of alarm record attributes
      //in which you are interested.
      //-------------------------------------------------------
-----
      AlarmRecordAttributeSet attrSet = new
AlarmRecordAttributeSet();
      attrSet.add(AlarmRecordAttribute.LOG_RECORD_ID);
      attrSet.add(AlarmRecordAttribute.PERCEIVED_SEVERITY);
      attrSet.add(AlarmRecordAttribute.PROBABLE_CAUSE);
      attrSet.add(AlarmRecordAttribute.CLEAR_STATE);
      attrSet.add(AlarmRecordAttribute.CLEAR_TIME);
      attrSet.add(AlarmRecordAttribute.ACK_TIME);
      attrSet.add(AlarmRecordAttribute.LOGGING_TIME);
      attrSet.add(AlarmRecordAttribute.EVENT_TYPE);
      attrSet.add(AlarmRecordAttribute.ACK_STATE);
      attrSet.add(AlarmRecordAttribute.MANAGED_OBJECT_INSTANCE);
      attrSet.add(AlarmRecordAttribute.ACK_OPERATOR);
      attrSet.add(AlarmRecordAttribute.ACK_TEXT);
      attrSet.add(AlarmRecordAttribute.CLEAR_OPERATOR);
      attrSet.add(AlarmRecordAttribute.CLEAR_TEXT);
      attrSet.add(AlarmRecordAttribute.ADDITIONAL_TEXT);
      attrSet.add(AlarmRecordAttribute.DISPLAY_OPERATOR);
      attrSet.add(AlarmRecordAttribute.DISPLAY_STATE);
      attrSet.add(AlarmRecordAttribute.DISPLAY_TIME);
      attrSet.add(AlarmRecordAttribute.EVENT_TIME);

      try {
        System.out.println("Connecting to " + args[1] + "...");

        //Create a Platform object based on the arguments passed
        //-----------------------------------------------------
```

CODE EXAMPLE 3-4 AlarmBatch *(Continued)*

```
        Platform platform =
                   new Platform(args[0],args[1] ,
args[2],args[3]);
        System.out.println("Platform instantiation complete ");

       //Instantiate a AlarmLog Object for a log named "AlarmLog"
       //-------------------------------------------------------
        System.out.println("AlarmLog instantiation");
        LogName logName = new LogName(args[1], "AlarmLog");
        AlarmLog log = new AlarmLog(platform, logName);

        //Create a query
        //---------------

       //Create array of perceived severity values to be used to
        //create a filter item
        EMSeverity[] sev = {
          EMSeverity.CRITICAL
        };

        // Create a FilterItem based on  perceivedSeverity
        FilterItem filterItem1 =
           new
FilterItem(AlarmRecordAttribute.PERCEIVED_SEVERITY,
            RelationCriteria.EQUAL, sev);

        Filter filter = new Filter(filterItem1);
        GenericQuery query = new GenericQuery(filter);

        //Test batch mechanism
        //--------------------

        // Create a listener object instance to handle to batch
events
        BatchListener batchList = new BatchListener("Critical");
        System.out.println("Getting critical alarms in batches");
        //Get alarms in batches of 5000 each
        int batchId =
           log.getAlarmsInBatches(query, 5000, batchList,
attrSet);
        System.out.println("Batch Id: " + batchId);

        //Stop batch
        //----------
        System.out.println("Stopping the batch");
        log.stopGetAlarmsInBatches(batchId);
```

CODE EXAMPLE 3-4 AlarmBatch *(Continued)*

```
      }
      catch(Exception e1) {
        e1.printStackTrace();
      }
  }

}

// Implements a listener object to handle batch events
//-------------------------------------------------------
class BatchListener implements AlarmsBatchListener {

  String id = null;
  public BatchListener(String id) {
    this.id = id;
  }

  // When a batch is received the following method is called
  public void batchReceived(int callId, AlarmRecord[] record) {
    System.out.println("BatchId: " + callId);
    System.out.println( "Number of alarms received: " +
record.length);

  }

  // When all batches have been received the following method is
called
  public void batchDone(int callId) {
    System.out.println("BatchId :" + callId);
    System.out.println("Batch complete");

  }
  private static final String sccsID =
           "@(#)AlarmBatch.java     1.5 98/10/30 Sun
Microsystems";

}
```

3.8.2 AlarmEvent

The following code example:

- Listens for alarm creation, deletion, and modification events.
- Prints the event data.

CODE EXAMPLE 3-5 AlarmEvent

```
/*
 * Copyright 10/30/98 Sun Microsystems, Inc. All Rights Reserved.
 */

/*
   AlarmEvent <servername> <mis-name> <username> <password>
   <servername> - is the machine name on which the server is
running.
   <mis-name> - is the machine name on which the mis is running.
   <username> - is the user login name.
   <password> - is the password of the user login.
*/

import com.sun.em.api.alarm.*;
import com.sun.em.api.common.*;
import java.util.Enumeration;
import com.sun.em.api.pmi.*;

public class AlarmEvent {

  static void usage() {
    System.err.println("Usage:");
    System.err.println("AlarmEvent  <servername> <mis-name>
<username>
                       <password> ");
    System.exit(-1);
  }

  public static void main(String[] args) {
    try {
      new AlarmEvent(args);
    }
    catch (Exception e) {
      e.printStackTrace();
    }

    System.out.println("Done.");
    System.exit(0);
```

CODE EXAMPLE 3-5 AlarmEvent *(Continued)*

```
  }

  public AlarmEvent(String[] args)
    throws AlarmException {

      Platform platform = null;
      if (args.length < 4)
        usage();

      try {

        //Connect to a platform
        //---------------------
        System.out.println("Connecting to " + args[1] + "...");
       platform = new Platform(args[0],args[1] , args[2],args[3]);
        System.out.println("Platform instantiation complete ");

       //Instantiate a AlarmLog Object for a log named "AlarmLog"
       //-------------------------------------------------------
        System.out.println("AlarmLog instantiation");
        LogName logName = new LogName(args[1], "AlarmLog");
        AlarmLog log = new AlarmLog(platform, logName);

      //Create an attribute set: The set of alarm record attributes
        //in which you are interested.
        //-------------------------------------------------------
----
        AlarmRecordAttributeSet attrSet = new
AlarmRecordAttributeSet();
        attrSet.add(AlarmRecordAttribute.LOG_RECORD_ID);
        attrSet.add(AlarmRecordAttribute.PERCEIVED_SEVERITY);
        attrSet.add(AlarmRecordAttribute.PROBABLE_CAUSE);
        attrSet.add(AlarmRecordAttribute.CLEAR_STATE);
        attrSet.add(AlarmRecordAttribute.CLEAR_TIME);
        attrSet.add(AlarmRecordAttribute.ACK_TIME);
        attrSet.add(AlarmRecordAttribute.ACK_STATE);
        attrSet.add(AlarmRecordAttribute.LOGGING_TIME);
        attrSet.add(AlarmRecordAttribute.EVENT_TYPE);
        attrSet.add(AlarmRecordAttribute.ACK_OPERATOR);
       attrSet.add(AlarmRecordAttribute.MANAGED_OBJECT_INSTANCE);

        //Listen for alarm related events
        //-------------------------------
        log.setEventAttrSet(attrSet);
        System.out.println("Listening for alarm events");
        log.addAlarmLogListener(new EventListener());
        Object o = new Object();
```

CODE EXAMPLE 3-5 AlarmEvent *(Continued)*

```
        synchronized(o) {
          try {
            o.wait();
          }
          catch (InterruptedException ex) {}
        }
      }
      catch(Exception e1) {
        e1.printStackTrace();
      }

  }

}

class EventListener implements AlarmLogListener {

  //Callback for alarm record creation
  //----------------------------------
  public void alarmRecordCreated(AlarmLogEvent event) {
    System.out.println("Received Alarm CREATION");
    try {
      //Print the event data
      //--------------------
      AlarmRecord alr1 = event.getAlarmRecord();
      System.out.println(alr1.toString());
    }
    catch (Exception e) {
      e.printStackTrace();
      System.exit(-1);
    }

  }

  //Callback for alarm record deletion
  //----------------------------------
  public void alarmRecordDeleted(AlarmLogEvent event) {
    System.out.println("Received Alarm DELETION");
    try {
      //Print the event data
      //--------------------
      AlarmRecord alr1 = event.getAlarmRecord();
      System.out.println(alr1.toString());
    }
    catch (Exception e) {
      e.printStackTrace();
      System.exit(-1);
```

CODE EXAMPLE 3-5 AlarmEvent *(Continued)*

```
    }
  }
  //Callback for alarm record modification
  //---------------------------------------
  public void alarmRecordModified(AlarmLogEvent event) {
    System.out.println("Received Alarm MODIFICATION");
    try {
      //Print the event data
      //--------------------
      AlarmRecord alr1 = event.getAlarmRecord();
      System.out.println(alr1.toString());
    }
    catch (Exception e) {
      e.printStackTrace();
      System.exit(-1);
    }

  }
  private static final String sccsID =
           "@(#)AlarmEvent.java     1.5 98/10/30 Sun
Microsystems";

}
```

3.8.3 AlarmClear

The following code example:

- Retrieves all alarms which have major or critical severity.
- Sets the display flag of the first alarm it retrieves.
- Acknowledges the first alarm it retrieves.
- Clears the first alarm it retrieves.

CODE EXAMPLE 3-6 AlarmClear

```
/*
 * Copyright 10/30/98 Sun Microsystems, Inc. All Rights Reserved.
 */

/*
   AlarmClear <servername> <mis-name> <username> <password>
   <servername> - is the machine name on which the server is
running.
   <mis-name> - is the machine name on which the mis is running.
   <username> - is the user login name.
   <password> - is the password of the user login.
*/

import com.sun.em.api.alarm.*;
import com.sun.em.api.common.*;
import java.util.Enumeration;
import java.util.Vector;
import com.sun.em.api.pmi.*;

public class AlarmClear {

  static void usage() {
    System.err.println("Usage:");
    System.err.println("AlarmClear  <servername> <mis-name>
<username> <password> ");
    System.exit(-1);
  }

  public static void main(String[] args) {
    try {
      new AlarmClear(args);
    }
    catch (Exception e) {
      e.printStackTrace();
    }

    System.out.println("Done.");
```

CODE EXAMPLE 3-6 AlarmClear *(Continued)*

```
    System.exit(0);

  }

  public AlarmClear(String[] args)
    throws AlarmException {

      if (args.length < 4)
        usage();

     //Create an attribute set: The set of alarm record attributes
in
      //which you are interested.
      //-------------------------------------------------------
-----
      AlarmRecordAttributeSet attrSet = new
AlarmRecordAttributeSet();
      attrSet.add(AlarmRecordAttribute.LOG_RECORD_ID);
      attrSet.add(AlarmRecordAttribute.PERCEIVED_SEVERITY);
      attrSet.add(AlarmRecordAttribute.PROBABLE_CAUSE);
      attrSet.add(AlarmRecordAttribute.CLEAR_STATE);
      attrSet.add(AlarmRecordAttribute.CLEAR_TIME);
      attrSet.add(AlarmRecordAttribute.CLEAR_TEXT);
      attrSet.add(AlarmRecordAttribute.CLEAR_OPERATOR);
      attrSet.add(AlarmRecordAttribute.ACK_STATE);
      attrSet.add(AlarmRecordAttribute.ACK_TIME);
      attrSet.add(AlarmRecordAttribute.ACK_TEXT);
      attrSet.add(AlarmRecordAttribute.LOGGING_TIME);
      attrSet.add(AlarmRecordAttribute.EVENT_TYPE);
      attrSet.add(AlarmRecordAttribute.MANAGED_OBJECT_INSTANCE);
      attrSet.add(AlarmRecordAttribute.LOG_NAME);
      attrSet.add(AlarmRecordAttribute.ADDITIONAL_TEXT);
      attrSet.add(AlarmRecordAttribute.DISPLAY_OPERATOR);
      attrSet.add(AlarmRecordAttribute.DISPLAY_STATE);
      attrSet.add(AlarmRecordAttribute.DISPLAY_TIME);
      attrSet.add(AlarmRecordAttribute.EVENT_TIME);

      try {
        //Create a query based on perceivedSeverity
        //-------------------------------
        EMSeverity[] sev = {
          EMSeverity.MAJOR,
          EMSeverity.CRITICAL
        };
        FilterItem filterItem1 =
           new
FilterItem(AlarmRecordAttribute.PERCEIVED_SEVERITY,
```

CODE EXAMPLE 3-6 AlarmClear *(Continued)*

```
            RelationCriteria.EQUAL, sev);

        Filter filter = new Filter(filterItem1);
        GenericQuery query = new GenericQuery(filter);
        //Connect to a platform
        //---------------------
        System.out.println("Connecting to " + args[1] + "...");
        Platform platform =
              new Platform(args[0],args[1] , args[2],args[3]);
        System.out.println("Platform instantiation complete ");

       //Instantiate a AlarmLog Object for a log named "AlarmLog"
      //-------------------------------------------------------
        System.out.println("AlarmLog instantiation");
        LogName logName = new LogName(args[1], "AlarmLog");
        AlarmLog log = new AlarmLog(platform, logName);

        //Now retrieve the count of alarms satisfying the query
        //-----------------------------------------------------
        System.out.println("\n Getting alarm Count for the
filter:\n");
        System.out.println(query.toString());
        int count = log.getAlarmCount(query);
        System.out.println("Count of Alarms: " + count);

      System.out.println("Getting alarms for the above filter\n");
        AlarmRecord[] alarms = log.getAlarms(query, attrSet);

        //Now print out the result
        //------------------------
        printAlarmRecord(alarms, attrSet);

        if (alarms.length == 0)
          {
          System.out.println("Stop the test as no alarms match the
                              filter");
            System.exit(-1);
          }

        try {
          //Get the log record Id of the first alarm record
retrieved.
          //-------------------------------------------------
-----
          AlarmRecordId alrmId =alarms[0].getLogRecordId();
          AlarmRecordId[] alrm = {alrmId};
```

CODE EXAMPLE 3-6 AlarmClear *(Continued)*

```
            System.out.println("\n Get cleared alarm \n");
            FilterItem filterItem2 =
                 new FilterItem(AlarmRecordAttribute.LOG_RECORD_ID,
                  RelationCriteria.EQUAL, alrm);

            Filter filter2 = new Filter(filterItem2);
            //Clear previous query
            //---------------------
            query.clear();
            query.addFilter(filter2);
            System.out.println(query.toString());
            System.out.println("Getting alarms for the filter: ");
            query.toString();
            alarms = log.getAlarms(query, attrSet);
            printAlarmRecord(alarms, attrSet);
            //Set the display flag
            //---------------------
            String[] str2 = {""};
            System.out.println("\n Set Display alarm \n");
            log.setDisplayAlarms(alrm, str2);
            //Acknowledge the alarm
            //----------------------
            System.out.println("\n Set Ack alarm \n");
            log.setAckAlarms(alrm, str2);
            //Clear the first alarm retrieved.
            //--------------------------------
            System.out.println("\n Clear the alarm");
            log.setClearAlarms(alrm, null);
          }
          catch (AlarmAttributeNotSetException e) {
            e.printStackTrace();
          }
        }
        catch(Exception e1) {
          e1.printStackTrace();
        }
    }
    private static void printAlarmRecord(AlarmRecord[] alarms,
                                  AlarmRecordAttributeSet attrSet)
      throws AlarmException
    {
      System.out.println("Received : " + alarms.length + "Alarms ");
      int ii;
      for (ii=0; ii<alarms.length; ii++)
        {
          System.out.println("Alarm number:" + ii );
          AlarmRecord alr1 = (AlarmRecord)alarms[ii];
```

CODE EXAMPLE 3-6 AlarmClear *(Continued)*

```
          System.out.println(alr1.toString());
        }
    }
    private static final String sccsID =
              "@(#)AlarmClear.java     1.5 98/10/30 Sun
Microsystems";
}
```

3.8.4 AlarmDelete

The following code example:

- Queries to retrieve alarms based on `perceivedSeverity` from a log.
- Deletes the first alarm it retrieves.

CODE EXAMPLE 3-7 AlarmDelete

```
/*
 * Copyright 10/30/98 Sun Microsystems, Inc. All Rights Reserved.
 */

/*
   AlarmDelete <servername> <mis-name> <username> <password>
   <servername> - is the machine name on which the server is
running.
   <mis-name> - is the machine name on which the mis is running.
   <username> - is the user login name.
   <password> - is the password of the user login.
*/

import com.sun.em.api.alarm.*;
import com.sun.em.api.common.*;
import com.sun.em.api.common.MOName;
import java.util.Enumeration;
import java.lang.reflect.Array;
import com.sun.em.api.pmi.*;
import java.util.*;
import java.lang.Integer;
import java.text.*;

public class AlarmDelete {

  static void usage() {
    System.err.println("Usage:");
    System.err.println("AlarmDelete  <servername> <mis-name>
<username> <password> ");
    System.exit(-1);
  }

  public static void main(String[] args) {
    try {
      new AlarmDelete(args);
    }
    catch (Exception e) {
      e.printStackTrace();
    }
```

CODE EXAMPLE 3-7 AlarmDelete *(Continued)*

```
    System.out.println("Done.");
    System.exit(0);

  }

  public AlarmDelete(String[] args)
    throws AlarmException {

      if (args.length < 4)
        usage();

     //Create an attribute set: The set of alarm record attributes
      //in which you are interested.
      //-------------------------------------------------------------
------
      AlarmRecordAttributeSet attrSet = new
AlarmRecordAttributeSet();
      attrSet.add(AlarmRecordAttribute.LOG_RECORD_ID);
      attrSet.add(AlarmRecordAttribute.PERCEIVED_SEVERITY);
      attrSet.add(AlarmRecordAttribute.PROBABLE_CAUSE);
      attrSet.add(AlarmRecordAttribute.CLEAR_STATE);
      attrSet.add(AlarmRecordAttribute.CLEAR_TIME);
      attrSet.add(AlarmRecordAttribute.CLEAR_TEXT);
      attrSet.add(AlarmRecordAttribute.CLEAR_OPERATOR);
      attrSet.add(AlarmRecordAttribute.ACK_STATE);
      attrSet.add(AlarmRecordAttribute.ACK_TIME);
      attrSet.add(AlarmRecordAttribute.ACK_TEXT);
      attrSet.add(AlarmRecordAttribute.ACK_OPERATOR);
      attrSet.add(AlarmRecordAttribute.LOGGING_TIME);
      attrSet.add(AlarmRecordAttribute.EVENT_TYPE);
      attrSet.add(AlarmRecordAttribute.MANAGED_OBJECT_INSTANCE);
      attrSet.add(AlarmRecordAttribute.LOG_NAME);
      attrSet.add(AlarmRecordAttribute.ADDITIONAL_TEXT);
      attrSet.add(AlarmRecordAttribute.DISPLAY_OPERATOR);
      attrSet.add(AlarmRecordAttribute.DISPLAY_STATE);
      attrSet.add(AlarmRecordAttribute.DISPLAY_TIME);
      attrSet.add(AlarmRecordAttribute.EVENT_TIME);

      try {
        //Create a query based on perceivedSeverity
        //----------------------------------------
        EMSeverity[] sev = {
          EMSeverity.MAJOR,
          EMSeverity.CRITICAL
        };
```

CODE EXAMPLE 3-7 AlarmDelete *(Continued)*

```
        FilterItem filterItem1 =
           new
FilterItem(AlarmRecordAttribute.PERCEIVED_SEVERITY,
            RelationCriteria.EQUAL, sev);

        Filter filter = new Filter(filterItem1);
        GenericQuery query = new GenericQuery(filter);

        //Connect to a platform
        //----------------------
        System.out.println("Connecting to " + args[1] + "...");
        Platform platform =
              new Platform(args[0],args[1] , args[2],args[3]);
        System.out.println("Platform instantiation complete ");

       //Instantiate a AlarmLog Object for a log named "AlarmLog"
      //-------------------------------------------------------
        System.out.println("AlarmLog instantiation");
        LogName logName = new LogName(args[1], "AlarmLog");
        AlarmLog log = new AlarmLog(platform, logName);

        //Get the alarm count based on the query
        //---------------------------------------
         System.out.println("\n Getting alarm Count for the
filter:\n");
        System.out.println(query.toString());
        int count = log.getAlarmCount(query);
        System.out.println("Count of Alarms: " + count);

        //Get alarms based on the query
        //-----------------------------
     System.out.println("Getting alarms for the above filter\n");
        AlarmRecord[] alarms = log.getAlarms(query, attrSet);

        //Now print out the result
        //------------------------
        printAlarmRecord(alarms, attrSet);

        if (alarms.length == 0)
          {
          System.out.println("Stop the test as no alarms match the
                              filter");
            System.exit(-1);
          }

        //Delete alarms
        //------------
```

CODE EXAMPLE 3-7 AlarmDelete *(Continued)*

```
        try {

        //Get the alarm Id of the first alarm and delete the alarm.
          //-------------------------------------------------------
----
          AlarmRecordId alarmRcdId = alarms[0].getLogRecordId();
          System.out.println("Delete the first alarm: " +
alarmRcdId.toString());
          AlarmRecordId[] alrmIdArray = {alarmRcdId};
          log.deleteAlarms(alrmIdArray);

          //Try to get it back to make sure
          //-------------------------------
          FilterItem fdelete =
               new FilterItem(AlarmRecordAttribute.LOG_RECORD_ID,
               RelationCriteria.EQUAL, alrmIdArray);
          Filter filterDel = new Filter(LogicalCriteria.AND,
fdelete);
          GenericQuery queryDel = new GenericQuery(filterDel);
         System.out.println("Try to get the alarm being deleted");
          System.out.println(queryDel.toString());
          AlarmRecord[] alrm = log.getAlarms(queryDel, attrSet);
          printAlarmRecord(alrm, attrSet);
        }
        catch (AlarmAttributeNotSetException e) {
          e.printStackTrace();
        }

      }
      catch (Exception e) {
        e.printStackTrace();
      }

  }

  private static void printAlarmRecord(AlarmRecord[] alarms,
                                AlarmRecordAttributeSet attrSet)
    throws AlarmException
  {
    System.out.println("Received : " + alarms.length + "Alarms ");
    int ii;
    for (ii=0; ii<alarms.length; ii++)
      {
        System.out.println("Alarm number:" + ii );
        AlarmRecord alr1 = (AlarmRecord)alarms[ii];
        System.out.println(alr1.toString());
      }
```

CODE EXAMPLE 3-7 AlarmDelete *(Continued)*

```
  }
  private static final String sccsID =
          "@(#)AlarmDelete.java    1.5 98/10/30 Sun
Microsystems";

}
```

3.8.5 AlarmQuery

The following code example:

- Queries to retrieve all alarms from a log.
- Queries to retrieve alarms based on perceivedSeverity from a log.
- Queries to retrieve alarms based on loggingTime.
- Gets alarm count based on a query.

CODE EXAMPLE 3-8 AlarmQuery

```
/*
 * Copyright 10/30/98 Sun Microsystems, Inc. All Rights Reserved.
 */

/*
   AlarmQuery <servername> <mis-name> <username> <password>
   <servername> - is the machine name on which the server is
running.
   <mis-name> - is the machine name on which the mis is running.
   <username> - is the user login name.
   <password> - is the password of the user login.
*/



import com.sun.em.api.alarm.*;
import com.sun.em.api.common.*;
import com.sun.em.api.common.MOName;
import java.util.Enumeration;
import java.lang.reflect.Array;
import com.sun.em.api.pmi.*;
import java.util.*;
import java.lang.Integer;
import java.text.*;

public class AlarmQuery {
```

CODE EXAMPLE 3-8 AlarmQuery *(Continued)*

```
  static void usage() {
    System.err.println("Usage:");
    System.err.println("AlarmQuery  <servername> <mis-name>
<username> <password> ");
    System.exit(-1);
  }

  public static void main(String[] args) {
    try {
      new AlarmQuery(args);
    }
    catch (Exception e) {
      e.printStackTrace();
    }

    System.out.println("Done.");
    System.exit(0);

  }

  public AlarmQuery(String[] args)
    throws AlarmException {

      if (args.length < 4)
        usage();

     //Create an attribute set: The set of alarm record attributes
      //in which you are interested.
      //-------------------------------------------------------
-----
      AlarmRecordAttributeSet attrSet = new
AlarmRecordAttributeSet();
      attrSet.add(AlarmRecordAttribute.LOG_RECORD_ID);
      attrSet.add(AlarmRecordAttribute.PERCEIVED_SEVERITY);
      attrSet.add(AlarmRecordAttribute.PROBABLE_CAUSE);
      attrSet.add(AlarmRecordAttribute.CLEAR_STATE);
      attrSet.add(AlarmRecordAttribute.CLEAR_TIME);
      attrSet.add(AlarmRecordAttribute.CLEAR_TEXT);
      attrSet.add(AlarmRecordAttribute.CLEAR_OPERATOR);
      attrSet.add(AlarmRecordAttribute.ACK_STATE);
      attrSet.add(AlarmRecordAttribute.ACK_TIME);
      attrSet.add(AlarmRecordAttribute.ACK_TEXT);
      attrSet.add(AlarmRecordAttribute.ACK_OPERATOR);
      attrSet.add(AlarmRecordAttribute.LOGGING_TIME);
      attrSet.add(AlarmRecordAttribute.EVENT_TYPE);
      attrSet.add(AlarmRecordAttribute.MANAGED_OBJECT_INSTANCE);
```

CODE EXAMPLE 3-8 AlarmQuery *(Continued)*

```
      attrSet.add(AlarmRecordAttribute.LOG_NAME);
      attrSet.add(AlarmRecordAttribute.ADDITIONAL_TEXT);
      attrSet.add(AlarmRecordAttribute.DISPLAY_OPERATOR);
      attrSet.add(AlarmRecordAttribute.DISPLAY_STATE);
      attrSet.add(AlarmRecordAttribute.DISPLAY_TIME);
      attrSet.add(AlarmRecordAttribute.EVENT_TIME);

      //Create array of perceived severity values
      try {
        // Create a FilterItem based on  perceivedSeverity
        //-------------------------------------------------
        EMSeverity[] sev = {
          EMSeverity.MAJOR,
          EMSeverity.CRITICAL
        };
        FilterItem filterItem1 =
           new
FilterItem(AlarmRecordAttribute.PERCEIVED_SEVERITY,
            RelationCriteria.EQUAL, sev);


        Filter filter = new Filter(filterItem1);

        //Connect to a platform
        //---------------------
        System.out.println("Connecting to " + args[1] + "...");
        // Create a Platform object based on the arguments passed
        Platform platform = new Platform(args[0],args[1] ,
args[2],args[3]);
        System.out.println("Platform instantiation complete ");

       //Instantiate a AlarmLog Object for a log named "AlarmLog"
      //-------------------------------------------------------
        System.out.println("AlarmLog instantiation");
        LogName logName = new LogName(args[1], "AlarmLog");
        AlarmLog log = new AlarmLog(platform, logName);

        //Get the total count of all existing alarms
        //------------------------------------------
        // Create a query with no filters
        GenericQuery query = new GenericQuery();
         System.out.println("\n Getting alarm Count for the
filter:\n");
        System.out.println(query.toString());
        int count = log.getAlarmCount(query);
        System.out.println("Count of Alarms: " + count);
```

CODE EXAMPLE 3-8 AlarmQuery *(Continued)*

```
        //Get alarms based on a query
        //---------------------------

        // Add the filter created earlier to the query to change
the query
        query.addFilter(filter);
         System.out.println("\n Getting alarm Count for the
filter:\n");
        System.out.println(query.toString());
        // Get the count of existing alarms based on the query
        count = log.getAlarmCount(query);
        System.out.println("Count of Alarms: " + count);

      System.out.println("Getting alarms for the above filter\n");
        //Get alarms based on the query
        AlarmRecord[] alarms = log.getAlarms(query, attrSet);

        //Now print out the result
        //------------------------
        printAlarmRecord(alarms, attrSet);

        if (alarms.length == 0)
          {
          System.out.println("Stop the test as no alarms match the
                              filter");
            System.exit(-1);
          }

        //Getting alarms created after a specific date
        //--------------------------------------------
        try {
          Date loggingTime = alarms[0].getLoggingTime();
        System.out.println("Getting alarm Count for alarms created
                      after or on:" + loggingTime.toString());

          FilterItem filterItem3 =
               new FilterItem(AlarmRecordAttribute.LOGGING_TIME,
            RelationCriteria.GREATER_THAN_OR_EQUAL, loggingTime);

          Filter filter1 = new Filter(filterItem3);
          //Add to the previous query
          query.addFilter(filter1);
          System.out.println(query.toString());
          count = log.getAlarmCount(query);
          System.out.println("Received: " + count +
                   " alarms matching the filter");
        }
```

CODE EXAMPLE 3-8 AlarmQuery *(Continued)*

```
        catch (AlarmAttributeNotSetException e) {
          e.printStackTrace();
        }

        //Getting alarms for perceived severity and clear State
        //-----------------------------------------------------
        FilterItem filterItem2 =
               new FilterItem(AlarmRecordAttribute.CLEAR_STATE,
                RelationCriteria.EQUAL, Boolean.TRUE);

        Filter filter2 = new Filter(filterItem2);
        query.addFilter(filter2);
        System.out.println("\n Getting alarm Count for the
filter:\n");
        System.out.println(query.toString());
        count = log.getAlarmCount(query);
        System.out.println("Count of Alarms: " + count);

      }
      catch (Exception e) {
        e.printStackTrace();
      }

  }

  private static void printAlarmRecord(AlarmRecord[] alarms,
                                AlarmRecordAttributeSet attrSet)
    throws AlarmException
  {
    System.out.println("Received : " + alarms.length + "Alarms ");
    int ii;
    for (ii=0; ii<alarms.length; ii++)
      {
        System.out.println("Alarm number:" + ii );
        AlarmRecord alr1 = (AlarmRecord)alarms[ii];
        System.out.println(alr1.toString());
      }

  }
  private static final String sccsID =
           "@(#)AlarmQuery.java     1.5 98/10/30 Sun
Microsystems";

}
```

CHAPTER 4

Using the Java Topology API

The Java Topology API defines the classes and methods which allow you to create applications that perform topology management operations without learning the details of the MIT naming tree. This chapter covers the important aspects of using the API and provides detailed examples. For more information about the API, refer to *Java API Reference* and *C++ API Reference*.

The following topics are covered in this chapter:

4.1 Overview

The Java Topology API allows you to create topology nodes to logically model managed objects in their management environments. Each topology node can correspond to one or more agents.

You can create applications using the Topology API. The following figure shows a GUI-based viewer which is an example of such an application. A GUI-based viewer, allows you to visualize interconnection among devices in a network.

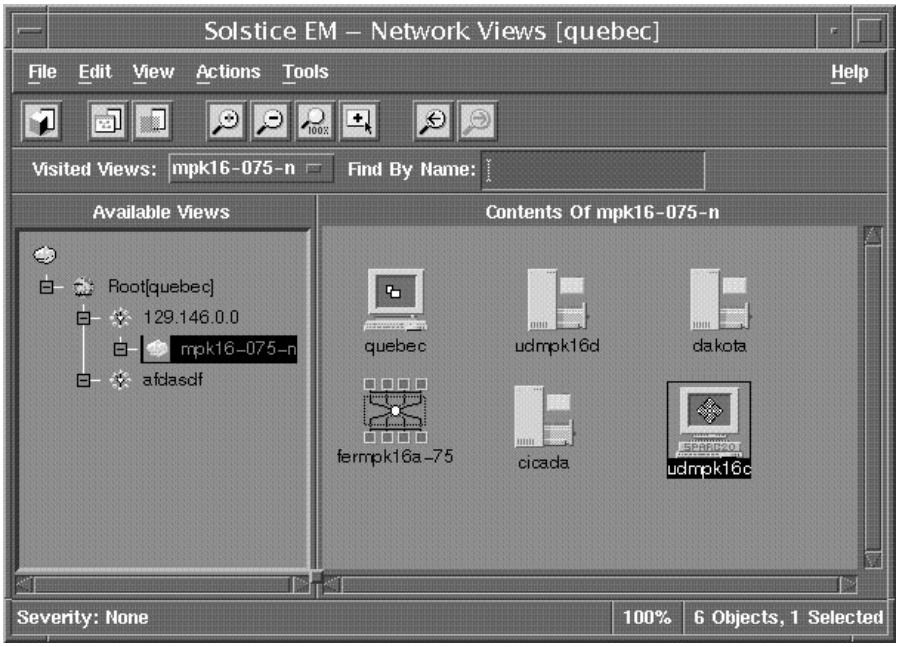

FIGURE 4-1 Sample Viewer Application

4.1.1 Topology Management Tasks

The Java Topology API allows you to perform the following topology management tasks:

- Node operations
- Type operations
- Agent operations

These tasks could be performed using the following main classes of the Java Topology API:

- The `EMTopoNode` class allows you to perform the different operations on MOIs.
- The `EMTopoType` class allows you to define node types.
- The `EMCmipAgent`, `EMRpcAgent`, and `EMSnmpAgent` classes allow you to perform agent operations.

4.2 Differences Between the C++ and Java Topology APIs

The Java Topology API is very similar to the C++ Topology API. For example, the main classes are still `EMTopoNode`, `EMTopoType`, `EMCmipAgent`, `EMSnmpAgent`, and `EMRpcAgent` in the Java Topology API. The other classes are helper classes. The following differences exist:

- The Java Topology API contains more classes than the C++ Topology API since almost all the `enum` and `struct` types in the C++ Topology API have classes as their equivalent in Java Topology API. For example, the `EMTopoNode` class in the C++ Topology API has an `enum` called `Attribute` that is represented as a class (`EMTopoNodeAttribute`) in the Java Topology API.
- The following methods of the `EMTopoPlatform` class in the C++ Topology API are available in the `EMTopoNode` class of the Java Topology API (instead of `EMTopoPlatform` class).
  - `find_nodes_by_name`
  - `find_nodes_by_type`
  - `find_nodes_by_managed_object`
  - `find_mos_by_nodes`
  - `load_nodes_in_view`

  These equivalent methods in the Java Topology API have the following names:

  - `findNodesByName`
  - `findNodesByType`
  - `findNodesByManagedObject`
  - `findMOsByNodes`
  - `loadNodesInView`

Note – These methods provide support for the actions `topoNodeGetByName`, `topoNodeGetByType`, `topoNodeGetByMO`, `topoGetMODataByNodeList`, and `topoNodeChildAttrsByView` that are supported by the `topoNodeDB` GDMO object class.

- In the C++ Topology API, the `EMPpcAgent` class has an attribute called `schemas`, but in the Java API, the same attribute is called `INFOS`.

For more detailed information about the C++ Topology API, especially on which attributes are mandatory and settable for each persistent object, refer to Chapter 8 “Topology API” in *C++ API Reference*. And for more detailed information about the Java Topology API, refer to Chapter 4 “Java Topology API” in *Java API Reference.*

4.3 Performing Node Operations

The `EMTopoNode` class represents a topology node, which is the unit of topology management in Solstice EM. It provides methods for creating, deleting, loading, changing, and comparing topology nodes.

Using the `EMTopoNode` class' access methods, you can get and set the name, topology type, parents, topology pathname, logical and geographical location, and associated managed objects and their corresponding CMIP, RPC, and SNMP agent objects among other attributes. The `EMTopoNode` class also provides an event listening mechanism to notify clients when a topology node has been created, deleted, or has had one or more attributes changed.

This section covers three types of node operations that you can perform using the methods of the `EMTopoNode` class:

- Creating nodes
- Destroying nodes
- Changing node attributes

For more information about the `EMTopoNode` class, refer to Chapter 4 "Java Topology API" in *Java API Reference* and Chapter 8 "Topology API" in C++ *API Reference*.

Note – Before performing node operations, you must instantiate the `EMTopoPlatform` class.

4.3.1 Creating Nodes

▼ To Create a New Topology Node in the MIS

1. **Set the mandatory attributes required for creating the node.**
2. **Use the** `createWithAllAttributes` **or** `createWithSomeAttributes` **method to create the object in the persistent store.**

Note – The `createWithAllAttributes` method only uses attributes that have been given a value. If the create method succeeds, the `DN` attribute is set with the unique identifier of the new object.

The following code example shows how to create a topology node.

CODE EXAMPLE 4-1 Creating a Topology Node

```
import com.sun.em.api.pmi.Platform;
import com.sun.em.api.topology.*;

public class CreateTopoNode
{
  static void usage() {
    System.err.println("Usage:");
   System.err.println("CreateTopoNode <jma-servername> <mis-name>
<username> <password> <node-name> <typename> <parentname>");
    System.exit(-1);
  }

  public static void main(String[] args) {
    if (args.length < 7)
      usage();
    try {
      //Create a Platform and EMTopoPlatform:
     Platform platform = new Platform(args[0], args[1], args[2],
args[3]);
      EMTopoPlatform topoPlatform = new EMTopoPlatform(platform);

     // Create an empty EMTopoNode in memory (not persistent yet)
given
      //EMTopoPlatform instance
      EMTopoNode node = new EMTopoNode(topoPlatform);

      //Set attribute values (still in cache)
      node.setName(args[4]);
      node.setTypeName(args[5]);
```

CODE EXAMPLE 4-1 Creating a Topology Node *(Continued)*

```
//To set mandatory attribute parents, find the parent's dn(s).
      // In this example, if there are more than one parent with
the given name, choose the first one.

      EMTopoNodeDn parentDn = null;
    EMTopoNodeDn[] dns = EMTopoNode.findNodesByName(topoPlatform,
args[6]);
      if (dns.length == 0) {
    System.err.println("No nodes found with name " + args[6]);
    System.exit(-1);
      } else {
    for (int i = 0; i < dns.length; i++) {
     // The node is qualified to be a parent only when it is a view.
      if (EMTopoNode.isView(topoPlatform,dns[i])) {
        parentDn = dns[0];
        break;
      }
    }
    if (parentDn == null) {
      System.err.println("None of the nodes named \"" + args[6] +
"\" are views!");
      System.err.println("Aborting...");
      System.exit(-1);
    }
      }
      EMTopoNodeDn parents[] = { parentDn };
      node.setParents(parents);

      // Make the call to MIS to create a persistent node.
      node.createWithAllAttributes();
    }
    catch (Exception e) {
      e.printStackTrace();
      System.exit(-1);
    }
    System.err.println("Created node \"" + args[4] + "\"");
    System.exit(-1);
  }
}
```

4.3.2 Loading Node Attributes

▼ To Get the Attribute Values of a Topology Node Object

1. **Set the `DN` identifier.**

2. **Use the `loadAllAttributes` or `loadSomeAttributes` method.**

   Once the attribute values are loaded, they stay cached within the `EMTopoNode` in the API's user process memory space and remain constant even if the values change in the MIS. See the following code example.

CODE EXAMPLE 4-2 Loading Node Attributes

```
import com.sun.em.api.pmi.Platform;
import com.sun.em.api.topology.*;

public class LoadTopoNode
{
  static void usage() {
    System.err.println("Usage:");
   System.err.println("ChangeTopoNode <jma-servername> <mis-name>
<username> <password> <node-name>");
    System.exit(-1);
  }

  public static void main(String[] args) {
    if (args.length < 5)
      usage();
    try {
      //Create a Platform and EMTopoPlatform:
     Platform platform = new Platform(args[0], args[1], args[2],
args[3]);
      EMTopoPlatform topoPlatform = new EMTopoPlatform(platform);

     //Find all the nodes with the name indicated by the 5th input
argument,
    EMTopoNodeDn[] dns = EMTopoNode.findNodesByName(topoPlatform,
args[4]);
      if (dns.length == 0) {
    System.err.println("No nodes found with name \"" + args[4] +
"\"");
    System.exit(-1);
      } else {
    System.err.println("Found " + dns.length + " node(s) with name
\"" + args[4] + "\"");
      }
```

CODE EXAMPLE 4-2 Loading Node Attributes *(Continued)*

```
        for (int i = 0; i < dns.length; i++) {
      EMTopoNode node = new EMTopoNode(topoPlatform, dns[i]);
      // Load the attributes from MIS
      node.loadAllAttributes();
      System.err.println(node.toString());
        }
      }
      catch (Exception e) {
        e.printStackTrace();
        System.exit(-1);
      }
      System.err.println("Done");
      System.exit(-1);
    }
  }
```

There are several static methods of `EMTopoNode` that load multiple nodes instead of just one node:

- `EMTopoNode.loadNodes`
- `EMTopoNode.loadNodesInBatches`
- `EMTopoNode.loadNodesInView`

Refer to Chapter 4 “Java Topology API” in *Java API Reference* for more information.

4.3.3 Changing Node Attributes

▼ To Set the Attribute Values Persistently in the MIS

1. **Set the `DN` attribute to identify the topology node.**
2. **Use the setter methods to change attribute values of the node in the cache.**
3. **Call either `storeAllAttributes` or `storeSomeAttributes` to commit these changes persistently in the MIS.**

Note – The `storeAllAttributes` method only stores attributes that have been given a value.

The following code example shows how to change the name of topology nodes.

CODE EXAMPLE 4-3 Changing Node Attributes

```
import com.sun.em.api.pmi.Platform;
import com.sun.em.api.topology.*;

public class ChangeTopoNode
{
  static void usage() {
    System.err.println("Usage:");
   System.err.println("ChangeTopoNode <jma-servername> <mis-name>
<username> <password> <old-name> <new-name>");
    System.exit(-1);
  }

  public static void main(String[] args) {
    if (args.length < 6)
      usage();
    try {
      //Create a Platform and EMTopoPlatform
     Platform platform = new Platform(args[0], args[1], args[2],
args[3]);
      EMTopoPlatform topoPlatform = new EMTopoPlatform(platform);

      //Find all nodes with the name indicated by the 5th input
argument
    EMTopoNodeDn[] dns = EMTopoNode.findNodesByName(topoPlatform,
args[4]);
      if (dns.length == 0) {
    System.err.println("No nodes found with name \"" + args[4] +
"\"");
```

CODE EXAMPLE 4-3 Changing Node Attributes *(Continued)*

```
    System.exit(-1);
      } else {
    System.err.println("Found " + dns.length + " node(s) with name
\"" + args[4] + "\"");
      }

      for (int i = 0; i < dns.length; i++) {
    EMTopoNode node = new EMTopoNode(topoPlatform, dns[i]);
    //Set the node name to the new name, this setting is still in
cache
    node.setName(args[5]);
    // we are ready to commit the change to MIS
    node.storeAllAttributes();
      }
    }
    catch (Exception e) {
      e.printStackTrace();
      System.exit(-1);
    }
    System.err.println("Renamed node(s) of the name \"" + args[4]
+ "\" to the new name \"" + args[5] + "\"");
    System.exit(-1);
  }
}
```

4.3.4 Destroying Nodes

▼ To Destroy a Topology Node

1. **Set the** `DN` **identifier.**
2. **Use the** `destroy` **method to delete the object from the MIS.**

Caution – This is a permanent, non-reversible operation.

The following code example shows how to destroy a node.

CODE EXAMPLE 4-4 Destroying Nodes

```
import com.sun.em.api.pmi.Platform;
import com.sun.em.api.topology.*;

public class DestroyTopoNode
{
  static void usage() {
    System.err.println("Usage:");
    System.err.println("DestroyTopoNode <jma-servername> <mis-
name> <username> <password> <node-name>");
    System.exit(-1);
  }

  public static void main(String[] args) {
    if (args.length < 5)
      usage();
    try {
      //Create a Platform and EMTopoPlatform
     Platform platform = new Platform(args[0], args[1], args[2],
args[3]);
      EMTopoPlatform topoPlatform = new EMTopoPlatform(platform);

      // Find all nodes with the name indicated by the 5th input
argument
    EMTopoNodeDn[] dns = EMTopoNode.findNodesByName(topoPlatform,
args[4]);
      if (dns.length == 0) {
    System.err.println("No nodes found with name \"" + args[4] +
"\"");
    System.exit(-1);
      } else {
    System.err.println("Found " + dns.length + " node(s) with name
\"" + args[4] + "\"");
```

CODE EXAMPLE 4-4 Destroying Nodes *(Continued)*

```
      }

      //Create an empty EMTopoNode in cache
      EMTopoNode node = new EMTopoNode(topoPlatform);
      // delete found nodes one by one
      for (int i = 0; i < dns.length; i++) {
    node.setDn(dns[i]);
    // Make the call to MIS to really destory the object in
persistent store
    node.destroy();
      }
    }
    catch (Exception e) {
      e.printStackTrace();
      System.exit(-1);
    }
    System.err.println("Deleted node(s) of name \"" + args[4] +
"\"");
    System.exit(-1);
  }
}
```

4.3.5 Listening to Node Events

Node events include object creation, object deletion, and attribute value change events.

▼ To Listen to Topology-Node-Related Events

1. **Write a class that implements** `EMTopoNodeListener` **and define the body of its prototype methods according to your application needs.**
2. **Register the listener with the** `EMTopoPlatform` **object in a relevant place in your program.**

The following code example shows how to listen to node events.

CODE EXAMPLE 4-5 Listening to Node Events

```
import com.sun.em.api.pmi.Platform;
import com.sun.em.api.topology.*;

public class ListenToTopoNodeEvent implements EMTopoNodeListener
{
  static void usage() {
```

CODE EXAMPLE 4-5 Listening to Node Events *(Continued)*

```
    System.err.println("Usage:");
    System.err.println("ListenToTopoNode <jma-servername> <mis-
name> <username> <password>");
    System.exit(-1);
  }

  // the following three method topotypes are in EMTopoNodeListener
  public void nodeCreated(EMTopoNodeEvent event) {
    // do something particular to your application
    System.err.println(event.toString());
  }
  public void nodeChanged(EMTopoNodeEvent event) {
    // do something particular to your application
    System.err.println(event.toString());
  }
  public void nodeDeleted(EMTopoNodeEvent event) {
    // do something particular to your application
    System.err.println(event.toString());
  }

  public static void main(String[] args) {
    if (args.length < 4)
      usage();
    try {
      // first create a Platform and EMTopoPlatform
     Platform platform = new Platform(args[0], args[1], args[2],
args[3]);
      EMTopoPlatform topoPlatform = new EMTopoPlatform(platform);

      // create a topo node event listener
     ListenToTopoNodeEvent listener = new ListenToTopoNodeEvent();

      // install/register the listener
      EMTopoNode.addEMTopoNodeListener(topoPlatform, listener);
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

There is another type of listener (`EMIndividualNodeListener`) that only listens to object creation and attribute value change events of a single topology node. `EMTopoNodeListener` listens to events of any topology node. Refer to Chapter 4 “Java Topology API” in *Java API Reference* for more information.

4.4 Performing Type Operations

Type operations can be performed using the methods of the `EMTopoType` class. Each instance of this class represents a topology type that is used to classify topology nodes. The topology types form a hierarchy with the following base types (and their subtypes):

- Array
- Bus
- Container
- Device
- Link
- Monitor
- Sun

This section covers the following four topology type operations:

- Creating types
- Loading types
- Changing types
- Destroying types

For more information about the `EMTopoType` class, refer to Chapter 4 “Java Topology API” in *Java API Reference*.

4.4.1 Creating Topology Types

▼ To Create a New Topology Type

1. **Instantiate the `EMTopoType` class in the cache by providing the following two parameters:**
   - `Platform`: The object that gives your application access to a particular MIS.
   - `EMTopoTypeDn`: The object that uniquely identifies a topology type out of the set of topology types.
2. **Set at least the two mandatory attributes.**
3. **Use the `createWithAllAttributes` or `createWithSomeAttributes` method to create the object in the persistent store.**

The following code example shows how to create a topology type.

CODE EXAMPLE 4-6 Creating Types

```
public static void createType(EMTopoPlatform topoPlatform, String
name,
                                  String baseTypeName)
        throws EMTopoException {

            if (topoPlatform == null) {
                System.err.println("topoPlatform is null");
            }
            if (name == null) {
                System.err.println("name is null");
            }
            if (baseTypeName == null) {
                System.err.println("baseTypeName is null");
            }

            EMTopoTypeDn dn =  new

EMTopoTypeDn(topoPlatform.getLocalSystemName(),name);
            EMTopoType type = new EMTopoType(topoPlatform,dn);
            type.setBaseType(baseTypeName);
            type.setLayerName(baseTypeName);

            try {
                type.createWithAllAttributes();
            }
            catch (Exception e) {
                e.printStackTrace();
                System.exit(-1);
            }
            System.err.println("Created type \""+name+"\"");

    }
```

4.4.2 Loading Topology Types

Use the `loadAllAttributes` or `loadSomeAttributes` method to load the attributes of an `EMTopoType` object as shown in the following code example.

CODE EXAMPLE 4-7 Loading Types

```
public static void loadType(EMTopoPlatform topoPlatform,String
name)
        throws EMTopoException {
            EMTopoTypeDn dn = (EMTopoTypeDn) new

EMTopoTypeDn(topoPlatform.getLocalSystemName(),name);
            EMTopoType type = new EMTopoType(topoPlatform,dn);
            try {
                type.loadAllAttributes();
            }
            catch (Exception e) {
                e.printStackTrace();
                System.exit(-1);
            }

            System.err.println(type.toString());
    }
```

4.4.3 Changing Topology Types

Use the `storeAllAttributes` or `storeSomeAttributes` method to change the attribute values of a topology type persistent object as shown in the following code example.

CODE EXAMPLE 4-8 Changing Topology Types

```
public static void changeType(EMTopoPlatform topoPlatform, String
typeName, String layerName) {
    try {
      EMTopoTypeDn dn = (EMTopoTypeDn) new
    EMTopoTypeDn(topoPlatform.getLocalSystemName(), typeName);
      EMTopoType type = new EMTopoType(topoPlatform,dn);

      // set the layer name, this setting is still in cache
      type.setLayerName(layerName);

      // we are ready to commit the change to MIS
      type.storeAllAttributes();
```

CODE EXAMPLE 4-8 Changing Topology Types *(Continued)*

```
    }
    catch (Exception e) {
      e.printStackTrace();
      System.exit(-1);
    }
  }
```

4.4.4 Destroying Topology Types

Use the `EMTopoType destroy` method to destroy topology types as shown in the following code example.

CODE EXAMPLE 4-9 Destroying Types

```
public static void destroyType(EMTopoPlatform topoPlatform, String
name)
        throws EMTopoException {

            EMTopoTypeDn dn = (EMTopoTypeDn) new

EMTopoTypeDn(topoPlatform.getLocalSystemName(),name);
            EMTopoType type = new EMTopoType(topoPlatform,dn);
            try {
                type.destroy();
            }
            catch (Exception e) {
                e.printStackTrace();
                System.exit(-1);
            }
            System.err.println("Deleted type \""+name+"\"");
    }
```

4.5 Performing Agent Operations

The Java Topology API allows you to perform operations on three types of agents as shown in the following table.

TABLE 4-1 Agent Operation Types

Operation Type	Description
SNMP	The Java Topology API provides the `EMSnmpAgent` class that represents the MIS object containing configuration information for an SNMP agent. This information includes the read and write community strings, supported MIBs, and the transport address.
CMIP	The Java Topology API provides the `EMCmipAgent` class representing the MIS object containing configuration information for CMIP agent. This information includes the CMIP MPA hostname and port number, list of managed objects DN, network SAP, transport selector, presentation selector, session selector, and application entity title (AET).
RPC	The Java Topology API provides the `EMRpcAgent` class that represents the MIS object that contains configuration information for an RPC agent. This information includes the read and write community strings and supported schemas.

This section covers the following four types of agent operations:

- Creating agents
- Loading agents
- Changing agents
- Destroying agents

For more information on the classes `EMSnmpAgent`, `EMCmipAgent`, and `EMRpcAgent`, refer to Chapter 4 “Java Topology API” in *Java API Reference*.

Note – The following sections are based on RPC sample code. The same concepts can be applied to CMIP and SNMP agents.

4.5.1 Creating Agents

▼ To Create a New RPC Agent

1. **Instantiate the** `EMRpcAgent` **class in the cache by providing the following two parameters:**
   - `Platform`: The object that gives your application access to a particular MIS.
   - `RPCAgent`: The object that uniquely identifies an RPC agent out of the set of RPC agent objects.
2. **Set the agent's attributes, which are mandatory (and others if necessary).**
3. **Use the** `createWithAllAttributes` **or** `createWithSomeAttributes` **methods to create the object in the persistent store.**

   The following code example shows how to create an RPC agent.

CODE EXAMPLE 4-10 Creating Agents

```
public static void createRpcAgent(
        EMTopoPlatform topoPlatform, String name,
        String getCommunityString, String setCommunityString,
        EMRpcAgentInfo[] infos) throws EMTopoException {

            EMRpcAgentDn dn =  new

EMRpcAgentDn(topoPlatform.getLocalSystemName(),name);
           EMRpcAgent rpcAgent = new EMRpcAgent(topoPlatform,dn);
            rpcAgent.setGetCommunityString(getCommunityString);
            rpcAgent.setSetCommunityString(setCommunityString);
            if (infos != null) {
                rpcAgent.setInfos(infos);
            } else {
                infos = new EMRpcAgentInfo[1];
                infos[0] = new EMRpcAgentInfo("RPC Proxy -
ping","granite");
                rpcAgent.setInfos(infos);
            }

rpcAgent.setAdministrativeState(EMAgentAdministrativeState.UNLOC
KED);
            try {
                rpcAgent.createWithAllAttributes();
            }
            catch (Exception e) {
                e.printStackTrace();
                System.exit(-1);
```

CODE EXAMPLE 4-10 Creating Agents *(Continued)*

```
            }
            System.err.println("Created rpcAgent \""+name+"\"");
    }
```

4.5.2 Loading Agents

Use the `loadAllAttributes` or `loadSomeAttributes` method to load the attributes of an `EMRpcAgent` object as shown in the following code example.

CODE EXAMPLE 4-11 Loading Agents

```
public static void loadRpcAgent(EMTopoPlatform
topoPlatform,String name)
        throws EMTopoException {
            EMRpcAgentDn dn = (EMRpcAgentDn) new

EMRpcAgentDn(topoPlatform.getLocalSystemName(),name);
           EMRpcAgent rpcAgent = new EMRpcAgent(topoPlatform,dn);
            try {
                rpcAgent.loadAllAttributes();
            }
            catch (Exception e) {
                e.printStackTrace();
                System.exit(-1);
            }
            System.err.println(rpcAgent.toString());
    }
}
```

4.5.3 Changing Agents

Use the `storeAllAttributes` or `storeSomeAttributes` method to change the attribute values of an agent's persistent object as shown in the following code example.

CODE EXAMPLE 4-12 Changing Agents

```
public static void changeAgent(EMTopoPlatform topoPlatform, String
agentName, EMAgentAdministrativeState adminState) {
    try {
      EMRpcAgentDn dn = (EMRpcAgentDn) new
    EMRpcAgentDn(topoPlatform.getLocalSystemName(), agentName);
      EMRpcAgent rpcAgent = new EMRpcAgent(topoPlatform,dn);
      // Set the administrative state, this setting is still in
cache
      rpcAgent.setAdministrativeState(adminState);
      // You are ready to commit the change to MIS
      rpcAgent.storeAllAttributes();
    }
    catch (Exception e) {
      e.printStackTrace();
      System.exit(-1);
    }
  }
```

4.5.4 Destroying Agents

Use the `destroy` method to destroy RPC agents as shown in the following code example.

CODE EXAMPLE 4-13 Destroying Agents

```
public static void destroyRpcAgent(EMTopoPlatform topoPlatform,
String name)
        throws EMTopoException {

            EMRpcAgentDn dn = (EMRpcAgentDn) new

EMRpcAgentDn(topoPlatform.getLocalSystemName(),name);
           EMRpcAgent rpcAgent = new EMRpcAgent(topoPlatform,dn);
            try {
                rpcAgent.destroy();
            }
            catch (Exception e) {
                e.printStackTrace();
                System.exit(-1);
            }
            System.err.println("Deleted rpcAgent \""+name+"\"");
    }
```

CHAPTER 5

Configuring the JDMK Agent/Behavior Service

The Java Dynamic Management Kit (JDMK) provides a set of Java classes and interfaces that enable you to develop network agents and services in Java. To manage JDMK agents by using Solstice Enterprise Manager (Solstice EM), you have to configure the JDMK agent/behavior service of Solstice EM.

This chapter explains how to configure the JDMK agent/behavior service of Solstice EM. It covers the following topics:

5.1 Overview

Configuring the JDMK agent/behavior service of Solstice EM makes your JDMK agents visible to the Java and C++ Solstice EM APIs.

Configuring the JDMK agent/behavior service involves:

- Optional: setting up the JDMK management protocol adaptor (MPA)
- Generating GDMO from Java classes
- Compiling and loading the generated GDMO files into the MDR
- Optional: configuring persistent `jdmkAgent` objects
- Optional: testing your agent with the MIS Objects tool

Note – Attribute value change events are not supported by JDMK agents. To obtain attribute value change events, use the agent developer interface that enables agents to send generic events through the framework.

5.1.1 Supported Versions of JDMK

Any JDMK agent that you want to manage by using Solstice EM must have been developed with version 4.0 of JDMK. Earlier versions of JDMK are not compatible with Solstice EM.

5.1.2 Prerequisites for Configuring the JDMK Agent/Behavior Service

Before you configure the JDMK agent/behavior service, ensure that:

- The JDMK m-beans that you want to manage have been developed. For more information on how to develop JDMK m-beans, refer to the documentation supplied with JDMK.
- The JDMK m-beans that you want to manage have been compiled.
- The `CLASSPATH` environment variable identifies the location of the compiled m-beans.

5.2 Setting Up the JDMK MPA

The JDMK MPA and its associated event forwarder perform protocol translation between the JDMK information model and the Solstice EM information model. This translation allows MIS client applications to access and receive events from JDMK agents.

5.2.1 Configuring the JDMK MPA

The JDMK MPA is pre-configured for you. To override its configuration options, edit the start-up script `/etc/rc2.d/S98jdmkmpa`. The following table describes the JDMK MPA configuration options.

TABLE 5-1 JDMK MPA Configuration Options

Option	Description
`EM_MIS_DEFAULT_HOST`	Preconfigured to the same machine on which the JDMK MPA package is installed
`EM_JDMK_MPA_DEFAULT_PORT`	Port number to be used by JDMK MPA Default: `5583`
`EM_JDMK_MPA_LOG_FILE`	Path and name of log file that stores debugging and diagnostic messages Default: `/var/opt/SUNWconn/em/debug/em_mpa_jdmk.log`

Note – If you change the JDMK MPA configuration options, you must stop and restart the JDMK MPA for your changes to take effect.

5.2.2 Starting and Stopping the JDMK MPA

If the JDMK component is installed, the JDMK MPA and the JDMK event forwarder start automatically when Solstice EM is started. If you need to start or stop the JDMK MPA and the JDMK event forwarder (for example, because you have changed its configuration), you can do so from the command line.

▼ To Start the JDMK MPA and the JDMK Event Forwarder

- **As root, type the following command:**

```
# /etc/rc2.d/S98jdmkmpa start
```

▼ To Stop the JDMK MPA and the JDMK Event Forwarder

- **As root, type the following command:**

```
# /etc/rc2.d/S98jdmkmpa stop
```

5.3 Generating GDMO From Java Classes

Agents developed by using JDMK use the Java information model. For your JDMK agents to be visible to your Solstice EM applications, the class definitions of these agents must be represented in the GDMO information model, which Solstice EM uses internally.

To generate a representation in GDMO of your Java class definitions, use the Java to GDMO compiler (`em_java2gdmo`).

5.3.1 Class Definition Conversions

▼ To Convert Java Interfaces to a GDMO Document

- **Type the following command:**

```
% em_java2gdmo [-rootoid rootOid -output directory -verbose] interfaceList
```

The arguments to `em_java2gdmo` are defined in the following table.

TABLE 5-2 `em_java2gdmo` Arguments

Argument	Meaning
`-rootoid` *rootOid*	Specifies the root of the object identifier (OID) that all GDMO objects are registered under. See Section 5.3.3 “Assignment of Object Identifiers” on page 5-6 for more information.
`-output` *directory*	Specifies the directory where the generated GDMO document file is written.
`-verbose`	Specifies that the Java to GDMO compiler displays warning messages and error messages during compilation. By default it displays only error messages.
interfaceList	A list specifying one or more compiled Java `MBean` interfaces that you want to represent in GDMO. Each interface name you specify may include the Java package prefix. Separate each class name in the list with a space. The compiled Java classes must be stored at a location identified by the `CLASSPATH` environment variable.

5.3.2 Generation of GDMO Documents

The Java to GDMO compiler converts all the Java `MBean` interfaces in a Java package into one GDMO document file and names the file as follows:

```
em_jdmk_package.gdmo
```

Where *package* is the full name of the Java package containing the Java `MBean` interfaces with each dot (.) changed to an underscore (_). If the classes are not contained in a Java package, the file is named `em_jdmk_unpackaged.gdmo`.

For example, the interfaces `SimpleMBean`, `InterfaceMBean` and `TroutMBean` in the Java package `com.sun.em.sample` would be mapped to a single GDMO file named `em_jdmk_com_sun_em_sample.gdmo` containing these three object classes. See the following figure.

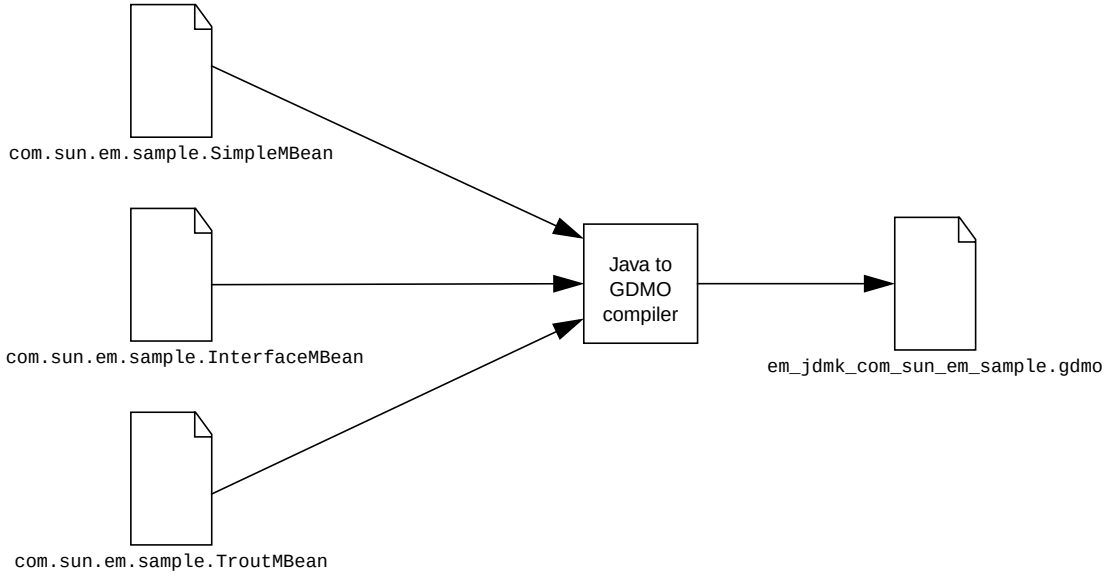

FIGURE 5-1 Java to GDMO Conversion

5.3.3 Assignment of Object Identifiers

The Java to GDMO compiler assigns all GDMO objects an object identifier (OID) of the form:

```
{ rootOid documentId objectType javaId }
```

The following table explains each part of this OID.

TABLE 5-3 Parts of an OID Assigned by `em_java2gdmo`

Part	Description
rootOid	The root of the OID as specified in the command you typed to start the Java to GDMO compiler. If you do not specify this root, it defaults to: `{iso(1) org(3) internet(1) private(4) enterprise(1) sun(42) products(2) management(2) em(2) jdmk(1)}`.
documentId	A hashed, unsigned, 32-bit value based on the document name. Not used if you specified the root of the OID in the command to start the compiler.
objectType	A number based on the type of object being registered as listed below: • `MANAGED OBJECT CLASS` object type OID: `6` • `ATTRIBUTE` object type OID: `7` • `ACTION` object type OID: `9` • `NOTIFICATION` object type OID: `5`
javaId	The ASCII values of the characters in the name of the object. Each character is separated from the one that precedes it by a period. For example, the class `coolClass` would translate to `99.111.111.108.67.108.97.115.115`.

5.3.4 Mapping Between Java Constructs and GDMO

Once a set of Java `MBean` interfaces have been identified for translation, each interface is examined through introspection to create a set of objects analogous to the GDMO objects that will be generated from the Java `MBean` interface.

The following table illustrates the mapping between Java classes, packages, methods, and GDMO objects.

TABLE 5-4 Mapping Between Java Constructs and GDMO Constructs

Java Construct	GDMO Analog Object
Package	`Module`
`MBean` Interface	`ManagedObjectClass`
Class derived from `com.sun.em.jdmk.event.EMEvent`	`Notification`
Getter, setter, or predicate method	`Attribute`
Other methods	`Action`

The first step in translation is to examine each of the Java `MBean` interface to be translated and decide if an interface will generate a `ManagedObjectClass` or `Notification`.

If the class extends `com.sun.em.jdmk.event.EMEvent`, a `Notification` is generated. Otherwise, a `ManagedObjectClass` is generated.

The attributes and actions of a managed object class are generated by examining each method in a Java `MBean` interface. Based on the method name, return type and parameter types, the Java to GDMO compiler does one of the following:

- Generates an `Action` object
- Generates an `Attribute` object

5.3.4.1 MODULE Construct

The GDMO MODULE is generated in the following form:

```
MODULE "EM JDMK JavaPackageName"
                    managedObjectClasses
                    notifications
END
```

The following table describes the keywords in italics.

TABLE 5-5 MODULE Keywords

Keyword	Description
JavaPackageName	The name of the Java package being translated. If the classes are not contained in a Java package, the document is named unpackaged EM JDMK.
managedObjectClasses	The GDMO equivalent for all managed object classes translated from the source Java package.
notifications	The GDMO equivalent for all notifications translated from the source Java package.

5.3.4.2 MANAGED OBJECT CLASS Construct

The GDMO MANAGED OBJECT CLASS is generated in the following form:

```
className MANAGED OBJECT CLASS
    DERIVED FROM "Rec. X.721 | ISO/IEC 10165-2 : 1992":top;
    CHARACTERIZED BY
    classNamePackage;
REGISTERED AS { classOid };

classNamePackage PACKAGE
    BEHAVIOUR interfaceBehaviour BEHAVIOUR DEFINED AS
        !This class represents java bean class className!;;
    ATTRIBUTES
                    -- naming attribute
                    classNameId GET,
                    -- other attributes
                    attributeList;
    ACTIONS
                    actionList;
    NOTIFICATIONS
                    "Rec. X.721 | ISO/IEC 10165-2 : 1992" :
            objectCreation,
                    "Rec. X.721 | ISO/IEC 10165-2 : 1992" :
            objectDeletion,
                    "Rec. X.721 | ISO/IEC 10165-2 : 1992" :
            attributeValueChange
                    notifications;
    ;

className-jdmkAgent NAME BINDING
    SUBORDINATE OBJECT CLASS className AND SUBCLASSES;
    NAMED BY SUPERIOR OBJECT CLASS "EM JDMK":jdmkAgent;
    WITH ATTRIBUTE classNameId;
    CREATE;
    DELETE;
REGISTERED AS { nameBindingOid };

classNameId ATTRIBUTE
    WITH ATTRIBUTE SYNTAX EM-JDMK.JavaString;
    MATCHES FOR      EQUALITY;
REGISTERED AS { attributeOid};
```

The following table describes the MANAGED OBJECT CLASS keywords in italics.

TABLE 5-6 MANAGED OBJECT CLASS Keywords

Keyword	Description
className	The Java MBean interface name with a lowercase first character and MBean string removed. A naming attribute *className*Id of type GraphicString and its name binding are generated. The name binding forces Java objects into a flat hierarchy under an instance of jdmkAgent.
actionList	The list of actions the managed object class contains. Each action is of the form *className-actionName*, where *className* is as defined above, and *actionName* is the name of the original Java method.
attributeList	The list of attributes contained in the class. Each attribute is of the form *className-attributeName*, where *className* is as defined above, and *attributeName* is the Java getter/setter method name, minus the get or set prefix, with the first character in upper case. Each attribute is defined as a GET or a GET-SET attribute as follows: • GET if only a getter is present in the original Java class. Predicates are always GET attributes. • GET-SET if the class contained both getters and setters. Attributes have no initial or default values and are *not* mandatory.
notifications	The list of all the custom notifications that a class supports and objectCreation, objectDeletion, and attributeValueChanged.
ClassOid	The OID of the managed object class. See Section 5.3.3 “Assignment of Object Identifiers” on page 5-6.
nameBindingOid	The OID of the generated name binding. See Section 5.3.3 “Assignment of Object Identifiers” on page 5-6.
attributeOid	The OID of the generated naming attribute. See Section 5.3.3 “Assignment of Object Identifiers” on page 5-6.

5.3.4.3 ATTRIBUTE Construct

The GDMO ATTRIBUTE is generated in the following form:

```
className-attributeName ATTRIBUTE
                    WITH ATTRIBUTE SYNTAX EM-JDMK.attributeType;
REGISTERED AS { oid };
```

The following table describes the keywords in italics.

TABLE 5-7 ATTRIBUTE Objects Keywords

Keyword	Description
className	The name of the managed object class to which the attribute belongs.
attributeName	The Java getter / setter method name, minus the `get` or `set` prefix, with the first character in upper case.
attributeType	The ASN.1 type based on the java type as described in Section 5.3.6 “Mapping JDMK Java Types to ASN.1” on page 5-15.
oid	The OID of the object. See Section 5.3.3 “Assignment of Object Identifiers” on page 5-6.

5.3.4.4 ACTION Construct

The GDMO ACTION is generated in the following form:

```
className-actionName ACTION
            BEHAVIOUR resetCountersBehaviour BEHAVIOUR DEFINED AS
            !action maps with actionName method!;;
            WITH INFORMATION SYNTAX EM-JDMK.paramType;
            WITH REPLY SYNTAX EM-JDMK.returnType;
REGISTERED AS { oid };
```

The following table describes the ACTION objects keywords.

TABLE 5-8 ACTION Objects Keywords

Keyword	Description
className	Name of the managed object class to which the action belongs.
actionName	Name of the original Java method.
attributeType	The ASN.1 type based on the Java method's attribute type as described in Section 5.3.6 "Mapping JDMK Java Types to ASN.1" on page 5-15.
returnType	The ASN.1 type based on the Java method's return type as described in Section 5.3.6 "Mapping JDMK Java Types to ASN.1" on page 5-15.
oid	The OID of the object. See Section 5.3.3 "Assignment of Object Identifiers" on page 5-6.

5.3.4.5 NOTIFICATION Construct

The GDMO NOTIFICATION is generated in the following form:

```
className Alarm NOTIFICATION
            BEHAVIOUR              className AlarmBehaviour;
            WITH INFORMATION SYNTAX
                Notification-ASN1Module.AlarmInfo
            AND ATTRIBUTE IDS
                probableCause      probableCause,
                perceivedSeverity  perceivedSeverity,
                additionalText     additionalText;
REGISTERED AS { oid };
```

The following table describes the keywords in italics.

TABLE 5-9 NOTIFICATION Objects Keywords

Keyword	Description
className	The name of the class from which the alarm is generated, with a lower case first character. The attribute values are: • probableCause: The OID of the class that generated the alarm. • perceivedSeverity: Indeterminate. • additionalText: A list of class attributes and their values in the form: {*attributeName*$_1$, *value*$_1$} {*attributeName*$_2$, *value*$_2$} ...
oid	The OID of the object. See Section 5.3.3 “Assignment of Object Identifiers” on page 5-6.

5.3.5 Mapping of M-Bean Object Names

The Java to GDMO compiler translates an m-bean's object name into a distinguished name. A translated distinguished name is in the following format:

```
jdmkAgentId="agentTableType=JDMK"/JavaClassNameId="ObjectName"
```

Where:

- *JavaClassName* is the Java class of the m-bean without the package prefix and with its initial letter set to lower case. It is separated from the GDMO document name by a colon.
- *ObjectName* is the object name of the m-bean. It is enclosed in quotes.

Note – The format of an m-bean's object name is defined in the documentation supplied with JDMK. Solstice EM imposes no additional restrictions on m-bean names.

The following table gives examples of the mapping of class names and m-bean names.

TABLE 5-10 Examples of Mapping Class Names and M-Bean Names

M-Bean Name:	`myDomain:name=le1`
Description:	Object of Java class `Interface` in package `em.sample` whose `name` attribute is `le1`.
GDMO MOC Name:	`"EM JDMK em.sample":interface`
Distinguished Name:	`jdmkAgentId="agentTableType=JDMK"/interfaceId="name=le1"`
M-Bean Name:	`myDomain:type=Interface`
Description:	Object of Java class `Interface` in package `em.sample.` The naming attribute is `null`. This object is a singleton and is the only allowed instance of its class.
GDMO MOC Name:	`"EM JDMK em.sample":interface`
Distinguished Name:	`jdmkAgentId="agentTableType=JDMK"/interfaceId="type=Interface"`
M-Bean Name:	`myDomain:name=le1,desc=fddi`
Description:	Object of Java class `Interface` in the default package (no package) whose `name` attribute is `le1` and `desc` attribute is `fddi`.
GDMO MOC Name:	`"unpackaged EM JDMK":interface`
Distinguished Name:	`jdmkAgentId="agentTableType=JDMK"/interfaceId="name=le1,desc=fddi"`

5.3.6 Mapping JDMK Java Types to ASN.1

The compiler supports only basic Java types. The following table lists the supported types with their ASN.1 counterparts. These types are legal return and attribute types for getters, setters, and predicate methods from which GDMO Analog objects are generated.

TABLE 5-11 Java to GDMO Type Translation

Java Class	ASN.1 Type
`String`	`Graphic String`
`Float`	`Real`
`Integer`	`INTEGER (-2147483648 .. 2147483647)`
`Long`	`INTEGER (-9223372036854775808 .. 9223372036854775807)`
`Boolean`	`BOOLEAN`
`Byte`	`OCTET (SIZE (1))`
`Short`	`INTEGER (-32768..32767)`
`void`	`NULL`
Class that implements `java.io.Serializable`[1]	`OCTET STRING`

1. Action parameters only.

5.3.7 Mapping Limitations

Although mapping is done automatically, there are a few limitations:

- Super classes and interfaces are *not* mapped.
- Only Java simple types and arrays of simple types are mapped. Other types are not mapped.
- Inner classes and anything private are ignored.
- Enums are not mapped.

5.4 Compiling and Loading the Generated GDMO Files Into the MDR

After you have generated a GDMO file representing your Java class definitions, you have to compile it and load it into the MetaData Repository (MDR) of Solstice EM.

To compile a GDMO file, use the `em_gdmo` command.

To load a GDMO file into the MDR, use the `em_compose_all` command.

To compile and load a GDMO file in a single operation, use the Load Data Definitions (LDD) tool.

For information on how to use `em_gdmo`, `em_compose_all`, and the LDD tool, refer to *Management Information System (MIS) Guide*.

Note – The JDMK MPA must be running when you load generated GDMO files into the MDR.

5.5 Configuring Persistent `jdmkAgent` Objects

To configure `jdmkAgent` objects that reside persistently in the MIS MIT, use the utility included with MIS `em_jdmk_config`. The `jdmkAgent` objects provide the MIS with the necessary information to communicate with configured JDMK agents.

Each `jdmkAgent` object contains three attributes. These attributes are described in the following table.

TABLE 5-12 `jdmkAgent` Object Attributes

Attribute	Description
`agentContainerId`	Names the `jdmkAgent` object. This name may be any string which the user chooses to represent a set of JDMK objects. For example, it may be the IP name where the JDMK agent resides in the network or it may be something which only represents a subset of the objects in the agent, such as topology objects. Think of this object as a named container to a set of JDMK objects.

TABLE 5-12 jdmkAgent Object Attributes *(Continued)*

Attribute	Description
agentAddr	A sequence containing the following: • agentAddr (default: value of agentContainerId) The IP name or IP address in dot format. • agentPort (default: 1099) Register the JDMK agent on this port. In your JDMK agent, you can specify what port the agent listens on. This feature allows you to run multiple agents or run other applications that need to use the default port. • agentAdaptor (default: RMI) A string containing the JDMK adapter that your agent is using. The only supported protocol is the remote method invocation (RMI) system. • agentDomain (default: defaultDomain) A string whose value is the Domain part of a JDMK object's name. You can only have one domain per jdmkAgent object. If your agent contains multiple domains, you will need to create a second jdmkAgent object with that domain.
supportedClasses	A list of Java class names including the package name. Note that the package name is the Java "Package.Class" name and not the GDMO class name.

5.5.1 Starting em_jdmk_config

The easiest way to configure a jdmkAgent object is to use the command-line: em_jdmk_config -config *yyy*, where *yyy* is the IP name or address of the machine on which the JDMK agent is running. This command goes out and interrogates the agent and configures the jdmkAgent object for you.

Note – The agent must be running and have the objects instantiated in order for this option to configure it. If there is an agent running with no objects, the jdmkAgent object will be configured, but the supportedClasses attribute will be empty.

The other way to configure an agent is to set the various em_jdmk_config options:

```
Usage: em_jdmk_config [-host] [-agent] [-help] [-get] [-delete]
[-set] [-create] [-dump] [-classes] [-info]
```

These options are described in the following table.

TABLE 5-13 `em_jdmk_config` Options

Option	Description
`-host` *mis*	MIS server name.
`-agent` *name*	Agent name for operation.
`-help`	Lists Help information.
`-get`	Prints the JDMK agent object specified with the `-agent` option.
`-delete`	Deletes the JDMK agent object specified with the `-agent` option.
`-set`	Sets the agent object (specified with the `-agent` option) with the information specified by the `-info` and `-classes` options.
`-create`	Creates the agent object (specified with the `-agent` option) with the information specified by the `-info` and `-classes` options.
`-dump`	Prints all JDMK agent objects.
`-info`	*ipaddr port adaptor domain* • *ipaddr*: Hostname or IP address on which the agent is running. • *port*: Port number on which the agent is listening for requests. • *adaptor*: The JDMK adaptor to be used for communication. Currently supports only RMI. • *domain*: The JDMK domain for this agent.
`-classes`	List of JDMK classes.

5.5.2 Configuration Examples

Before running the examples, make sure that the `CLASSPATH` environment variable identifies the location of the compiled example class files.

This section contains five configuration examples.

- The following example automatically configures the JDMK agent running on the host `titleist` using all default parameters.

Note – The agent *must* already be running with at least one object from each Java class.

```
% em_jdmk_config -config titleist
```

- The following example configures a JDMK agent running on `titleist` (using all defaults) with `Java classes em.sample.Interface` and `em.sample.Sample`:

```
% em_jdmk_config -create -agent titleist -classes
em.sample.Interface em.sample.Sample
```

- The following example prints out all configured agents:

```
% em_jdmk_config -dump
```

- The following example deletes the `jdmkAgent` object `titleist`. This example does not delete any objects on the JDMK agent. It only deletes the `jdmkAgent` object in the MIS that represents the JDMK agent:

```
% em_jdmk_config -delete -agent titleist
```

- The following example sets the supported classes on the `jdmkAgent` object `titleist`:

```
% em_jdmk_config -set -agent titleist -classes em.sample.NewClass
```

5.6 Testing Your Agent With the MIS Objects Tool

This section describes how to test your agents with the MIS Objects tool through an example. This example is based on the following three Java test files included with the JDMK example programs:

- `SimpleStandard.java` (Java `MBean` Class)

- `SimpleStandardMBean.java` (Java `MBean` Interface)
- `MinimalAgent.java` (JDMK Agent)

Note – For simplicity, the Java classes will be in the default package.

Once you have created your `jdmkAgent` object, you will be able to see it using the MIS Objects tool.

All the `jdmkAgent` objects may be found under the `/systemId="host"/agentTableType=JDMK` object. All the objects under each `jdmkAgent` object are the objects on that particular JDMK agent.

To enable the Solstice EM JDMK MPA to locate your JDMK classes, ensure that the `CLASSPATH` environment variable identifies where they are located.

By default, the JDMK MPA searches the following directories and jar files for Java classes:

- `/var/tmp/jdmk`
- *installdir*`/SUNWjdmk/jdmk4.2/1.2/lib/rt.jar`
- *installdir*`/SUNWconn/em/classes`

To ensure that the JDMK MPA can locate the sample Java classes, you can do one of two things:

- Store the classes in the `/var/tmp/jdmk` directory or create a link to it.
- Set the root `CLASSPATH` to include your specific `CLASSPATH` and restart the JDMK MPA.

```
# /etc/rc2.d/S98jdmkmpa stop
# setenv CLASSPATH yourpath
# /etc/rc2.d/S98jdmkmpa start
```

▼ To Perform the Sample Test

Before you perform the sample test, ensure that the `CLASSPATH` environment variable identifies where the JDMK classes and sample classes are located.

1. **Create a directory** `/var/tmp/jdmk`**.**
2. **Copy** `SimpleStandard.java, SimpleStandardMBbean.java` **and** `MinimalAgent.java` **to** `/var/tmp/jdmk`**.**
3. **Go to the** `/var/tmp/jdmk` **directory.**

4. **Compile the sample Java classes:**

```
% javac *.java
```

5. **Convert Java to GDMO.**

```
% em_java2gdmo SimpleStandardMbean
```

`em_jdmk_unpackaged.gdmo` file is generated by this command.

6. **Load the GDMO into the MIS.**

   a. **As root, start the Load Data Definitions tool.**

   b. **Go to** `/var/tmp/jdmk` **and load the file** `em_jdmk_unpackaged.gdmo`**.**

7. **Compile** `MinimalAgent`**.**

   Add the following lines in the MinimalAgent.java file to load the `SimpleStandardMBean` and compile the MinimalAgent.java file.

```
System.out.println("\nCreate SimpleStandard");
ObjectName objName = new
ObjectName("DefaultDomain:type=SimpleStandard");
ObjectInstance simple = server.createMBean("SimpleStandard",
objName);
```

```
% javac MinimalAgent.java
```

8. **Start the JDMK agent.**

```
% java MinimalAgent
```

9. **Run the following configuration command:**

```
% em_jdmk_config -create -agent yourhostname -info yourhostname 1099 \
name=RmiConnectorServer DefaultDomain -classes SimpleStandard
```

10. **Run the following configuration commands:**

```
% em_jdmk_config -dump
% em_jdmk_config -get -agent yourhostname
```

11. **Start up the MIS Objects tools from the Administration panel.**

12. **Browse down the** `agentTableType=JDMK` **part of the tree and look for your agent object.**

13. **Browse down your agent object and you should see the** `SimpleStandardMBean` **in the JDMK agent.**

14. **Set the** `SimpleStandard-State` **attribute of the** `SimpleStandardMBean` **object.**

15. **Get the** `SimpleStandard-State` **attribute of the** `SimpleStandardMBean` **object and verify that it has changed.**

16. **Select the same** `SimpleStandardMBean` **object and execute the** `SimpleStandard-reset` **action.**

    The action takes a NULL string. The `SimpleStandard` bean resets the state.

17. **Perform a** `get` **action and verify that the counters were reset.**

5.7 Sample Java Files

This section explains how to convert the sample programs. It also gives you the definition of a sample Java class, `SimpleStandard`, in CODE EXAMPLE 5-1 and its equivalent GDMO class in CODE EXAMPLE 5-3 as generated by the Java to GDMO compiler.

5.7.1 Converting Sample Java Classes Into GDMO

The following example shows you how to convert `SimpleStandardMBean.class` into GDMO.

▼ To Convert Sample Java Classes

1. **Run the** `SimpleStandardMBean.class` **file through the compiler using the following command line:**

```
% em_java2gdmo SimpleStandardMBean
```

The compiler generates one file: `em_jdmk_unpackaged.gdmo`. This file contains the GDMO document unpackaged `EM JDMK SimpleStandard` which contains the GDMO classes.

Note – If the `SimpleStandardMBean` class belongs to a package for example, `em.sample` then the resulting document is called "`EM JDMK em.sample`".

---

2. **Load the converted files into the MIS by using the Load Data Definitions tool or by typing the following command:**

```
% em_gdmo -host yourhost -file em_jdmk_em_sample.gdmo
```

5.7.2 Listings of the Sample Java Classes

The following code example contains a sample JDMK Agent.

CODE EXAMPLE 5-1 `MinimalAgent.java`

```
import javax.management.ObjectName;
import javax.management.ObjectInstance;
import javax.management.MBeanServer;
import javax.management.MBeanServerFactory;

public class MinimalAgent {

    public static void main(String[] args) {

    // Instantiate the MBean server
    System.out.println("\nCreate the MBean server");
```

CODE EXAMPLE 5-1 MinimalAgent.java *(Continued)*

```
import javax.management.ObjectName;
    MBeanServer server = MBeanServerFactory.createMBeanServer();

    // Create and start in the MBean server:
    //    - an HTML protocol adaptor
    //    - an HTTP connector server
    //    - an RMI connector server
    try {
        com.sun.jdmk.Trace.parseTraceProperties();
        System.out.println("\nCreate and start an HTML protocol
adaptor");
        ObjectInstance html =
server.createMBean("com.sun.jdmk.comm.HtmlAdaptorServer", null);
        server.invoke(html.getObjectName(), "start", new
Object[0], new String[0]);

        System.out.println("\nCreate and start an HTTP connector
server");
        ObjectInstance http =
server.createMBean("com.sun.jdmk.comm.HttpConnectorServer",
null);
        server.invoke(http.getObjectName(), "start", new
Object[0], new String[0]);

        System.out.println("\nCreate and start an RMI connector
server");
        ObjectInstance rmi =
server.createMBean("com.sun.jdmk.comm.RmiConnectorServer", null);
       server.invoke(rmi.getObjectName(), "start", new Object[0],
new String[0]);


        System.out.println("\nCreate SimpleStandard");
        ObjectName objName = new
ObjectName("DefaultDomain:type=SimpleStandard");
        ObjectInstance simple =
server.createMBean("SimpleStandard", objName);

    } catch(Exception e) {
        e.printStackTrace();
        return;
    }

    System.out.println("\nNow, you can point your browser to
http://localhost:8082/");
    System.out.println("or start your client application to
connect to this agent.\n");
```

CODE EXAMPLE 5-1 `MinimalAgent.java` *(Continued)*

```
import javax.management.ObjectName;
    }
}
```

The following code example contains a sample MBean interface.

CODE EXAMPLE 5-2 SimpleStandardMBean.java

```
/**
 * This is the management interface explicitly defined for the
"SimpleStandard" standard MBean.
 * The "SimpleStandard" standard MBean implements this interface
 * in order to be manageable through a JMX agent.
 *
 * The "SimpleStandardMBean" interface shows how to expose for
management:
 * - a read/write attribute (named "State") through its getter and
setter methods,
 * - a read-only attribute (named "NbChanges") through its getter
method,
 * - an operation (named "reset").
 */
public interface SimpleStandardMBean {

    /**
     * Getter: set the "State" attribute of the "SimpleStandard"
standard MBean.
     *
     * @return the current value of the "State" attribute.
     */
    public String getState() ;

    /**
     * Setter: set the "State" attribute of the "SimpleStandard"
standard MBean.
     *
     * @param <VAR>s</VAR> the new value of the "State" attribute.
     */
    public void setState(String s) ;

    /**
    * Getter: get the "NbChanges" attribute of the "SimpleStandard"
standard MBean.
     *
     * @return the current value of the "NbChanges" attribute.
     */
    public Integer getNbChanges() ;

    /**
     * Operation: reset to their initial values the "State" and
"NbChanges"
```

CODE EXAMPLE 5-2 SimpleStandardMBean.java *(Continued)*

```
/**
     * attributes of the "SimpleStandard" standard MBean.
     */
    public void reset() ;
}
```

The following code example contains a sample MBean implementation.

CODE EXAMPLE 5-3 SimpleStandard.java

```
/**
 * Simple definition of a standard MBean, named "SimpleStandard".
 *
 * The "SimpleStandard" standard MBean shows how to expose
attributes and
 * operations for management by implementing its corresponding
 * "SimpleStandardMBean" management interface.
 *
 * This MBean has two attributes and one operation exposed
 * for management by a JMX agent:
 *      - the read/write "State" attribute,
 *      - the read only "NbChanges" attribute,
 *  - the "reset()" operation.
 *
 * This object also has one property and one method not exposed
 * for management by a JMX agent:
 *  - the "NbResets" property,
 *  - the "getNbResets()" method.
 */

public class SimpleStandard implements SimpleStandardMBean {

    /*
     * --------------------------------------------------
     * CONSTRUCTORS
     * --------------------------------------------------
     */

   /* "SimpleStandard" does not provide any specific constructors.
     * However, "SimpleStandard" is JMX compliant with regards to
    * contructors because the default contructor SimpleStandard()
     * provided by the Java compiler is public.
     */
```

CODE EXAMPLE 5-3 SimpleStandard.java *(Continued)*

```
/**


    /*
     * -------------------------------------------------
     * IMPLEMENTATION OF THE SimpleStandardMBean INTERFACE
     * -------------------------------------------------
     */

    /**
     * Getter: get the "State" attribute of the "SimpleStandard"
standard MBean.
     *
     * @return the current value of the "State" attribute.
     */
    public String getState() {
        return state;
    }

    /**
     * Setter: set the "State" attribute of the "SimpleStandard"
standard MBean.
     *
     * @param <VAR>s</VAR> the new value of the "State" attribute.
     */
    public void setState(String s) {
        state = s;
        nbChanges++;
    }

    /**
    * Getter: get the "NbChanges" attribute of the "SimpleStandard"
standard MBean.
     *
     * @return the current value of the "NbChanges" attribute.
     */
    public Integer getNbChanges() {
        return new Integer(nbChanges);
    }

    /**
     * Operation: reset to their initial values the "State" and
"NbChanges"
     * attributes of the "SimpleStandard" standard MBean.
     */
    public void reset() {
```

CODE EXAMPLE 5-3 SimpleStandard.java *(Continued)*

```
/**
    state = "initial state";
        nbChanges = 0;
    nbResets++;
    }


    /*
     * ------------------------------------------------------
     * METHOD NOT EXPOSED FOR MANAGEMENT BY A JMX AGENT
     * ------------------------------------------------------
     */

    /**
     * Return the "NbResets" property.
     * This method is not a Getter in the JMX sense because
     * it is not exposed in the "SimpleStandardMBean" interface.
     *
     * @return the current value of the "NbResets" property.
     */
    public Integer getNbResets() {
    return new Integer(nbResets);
    }


    /*
     * ------------------------------------------------------
     * ATTRIBUTES ACCESSIBLE FOR MANAGEMENT BY A JMX AGENT
     * ------------------------------------------------------
     */

    private Stringstate = "initial state";
    private intnbChanges = 0;


    /*
     * ------------------------------------------------------
     * PROPERTY NOT ACCESSIBLE FOR MANAGEMENT BY A JMX AGENT
     * ------------------------------------------------------
     */

    private intnbResets = 0;
}
```

The following code example contains the em_jdmk_unpackaged.gdmo file.

CODE EXAMPLE 5-4 em_jdmk_unpackaged.gdmo

```
MODULE "unpackaged EM JDMK"

simpleStandard MANAGED OBJECT CLASS
    DERIVED FROM "Rec. X.721 | ISO/IEC 10165-2 : 1992":top;
    CHARACTERIZED BY
        simpleStandardPackage;
REGISTERED AS { em-jdmk 162229001 6 115 105 109 112 108 101 83 116
97 110 100 97 114 100 };

simpleStandardPackage PACKAGE
    BEHAVIOUR simpleStandardBehaviour BEHAVIOUR DEFINED AS
        !This class represents the Java Bean
classSimpleStandardMBean !;;
    ATTRIBUTES
        simpleStandardIdGET,
        simpleStandard-NbChangesGET,
        simpleStandard-StateGET-REPLACE;

    ACTIONS
        simpleStandard-reset;

    NOTIFICATIONS
        "Rec. X.721 | ISO/IEC 10165-2 : 1992" :
            objectCreation,
        "Rec. X.721 | ISO/IEC 10165-2 : 1992" :
            objectDeletion,
        "Rec. X.721 | ISO/IEC 10165-2 : 1992" :
            attributeValueChange;

    ;

simpleStandard-jdmkAgent NAME BINDING
    SUBORDINATE OBJECT CLASS simpleStandard AND SUBCLASSES;
    NAMED BY SUPERIOR OBJECT CLASS "EM MPA JDMK":jdmkAgent;
    WITH ATTRIBUTE simpleStandardId;
    CREATE;
    DELETE;
REGISTERED AS { em-jdmk 162229001 1 115 105 109 112 108 101 83 116
97 110 100 97 114 100 45 106 100 109 107 65 103 101 110 116 };

simpleStandardId ATTRIBUTE
    WITH ATTRIBUTE SYNTAX EM-JDMK.JavaString;
    MATCHES FOR      EQUALITY;
```

CODE EXAMPLE 5-4 em_jdmk_unpackaged.gdmo *(Continued)*

```
MODULE "unpackaged EM JDMK"
REGISTERED AS { em-jdmk 162229001 7 115 105 109 112 108 101 83 116
97 110 100 97 114 100 73 100 };

simpleStandard-NbChanges ATTRIBUTE
    WITH ATTRIBUTE SYNTAX EM-JDMK.JavaInteger;
REGISTERED AS { em-jdmk 162229001 7 115 105 109 112 108 101 83 116
97 110 100 97 114 100 45 78 98 67 104 97 110 103 101 115 };

simpleStandard-State ATTRIBUTE
    WITH ATTRIBUTE SYNTAX EM-JDMK.JavaString;
REGISTERED AS { em-jdmk 162229001 7 115 105 109 112 108 101 83 116
97 110 100 97 114 100 45 83 116 97 116 101 };

simpleStandard-reset ACTION
    BEHAVIOUR simpleStandard-resetBehaviour BEHAVIOUR DEFINED AS
        !action derived from SimpleStandardMBean.reset(...) Java
method!;;
    WITH INFORMATION SYNTAX EM-JDMK.JavaVoid;
    WITH REPLY SYNTAX EM-JDMK.JavaVoid;
REGISTERED AS { em-jdmk 162229001 9 115 105 109 112 108 101 83 116
97 110 100 97 114 100 45 114 101 115 101 116 };

END
```

APPENDIX **A**

Using the Java Alarm and Topology APIs Together

In some scenarios, it is useful to use the Java Alarm and Java Topology APIs together. For example, you may want to provide visual representation of alarms. This can be done by using the Java Topology API to create the topology node representation and the Alarm API to highlight the affected nodes.

This appendix provides a sample program that shows you how to use the Topology and Alarm APIs together (see the following code example). The program gets all the alarms pertaining to the specified node.

CODE EXAMPLE A-1 `GetAlarmsForNode.java`

```
 * Copyright 10/30/98 Sun Microsystems, Inc. All Rights Reserved.
 */
import com.sun.em.api.common.*;
import com.sun.em.api.topology.*;
import com.sun.em.api.alarm.*;
import com.sun.em.api.pmi.Platform;

/*
  GetAlarmsForNode <servername> <mis-name> <username> <password>
<name>
  <servername> - is the machine name on which the server is running.
   <mis-name> - is the machine name on which the mis is running.
   <username> - is the user login name.
   <password> - is the password of the user login.
   <name>     - is the name of the node.

 */

public class GetAlarmsForNode
{
```

CODE EXAMPLE A-1 GetAlarmsForNode.java *(Continued)*

```
  static void usage() {
   System.err.println("Usage:\n"+ "GetAlarmsForNode <servername>
                      <mis-name> <username> <password> name");
   System.err.println("\t-Run the example with <servername> as the
        remote server and <misname> \n\t where EM mis is
running.");
    System.err.println("\t name = Node name ");
    System.exit(-1);
  }

  public static void main(String[] args) {
    if (args.length < 5) {
      usage();
    }

    Platform platform = null;
    try {

      //Instantiate the Platform object
      platform = new Platform(args[0],args[1] , args[2],args[3]);

      //Instantiate a AlarmLog Object for a log named "AlarmLog"
      LogName logName = new LogName(args[1], "AlarmLog");
      System.out.println("AlarmLog instantiation");
      AlarmLog log = new AlarmLog(platform, logName);

      //Instantiate the EMTopoPlatform
      EMTopoPlatform topoPlatform = new EMTopoPlatform(platform);

      //Create a list of toponode ids to be used to retrieve the
     //managed objects. This is a list since there could be more
      //than one node with same name.
      EMTopoNodeDn dns[] = null;
      dns = EMTopoNode.findNodesByName(topoPlatform,args[4]);
      int ids[] = new int[dns.length];
      for(int k=0; k < dns.length; k++)
        ids[k] = dns[k].getUniqueId(); // get the toponode id

     //Get all the managed object names for the given toponode ids
      MOName[] monames = EMTopoNode.findMOsByNodes(topoPlatform,
                      args[1] , ids, false);

      //Create a query object to get all alarms for the list of
      //managed objects specified.
```

CODE EXAMPLE A-1 GetAlarmsForNode.java *(Continued)*

```
      FilterItem filterItem =
       new
FilterItem(AlarmRecordAttribute.MANAGED_OBJECT_INSTANCE,
        RelationCriteria.EQUAL, monames);

      Filter filter = new Filter(filterItem);
      GenericQuery query = new GenericQuery(filter);

     //Create an attribute set: The set of alarm record attributes
      //in which you are interested.
      //-------------------------------------------------------
----
      AlarmRecordAttributeSet attrSet = new
AlarmRecordAttributeSet();
      attrSet.add(AlarmRecordAttribute.LOG_RECORD_ID);
      attrSet.add(AlarmRecordAttribute.PERCEIVED_SEVERITY);
      attrSet.add(AlarmRecordAttribute.PROBABLE_CAUSE);
      attrSet.add(AlarmRecordAttribute.CLEAR_STATE);
      attrSet.add(AlarmRecordAttribute.CLEAR_TIME);
      attrSet.add(AlarmRecordAttribute.ACK_TIME);
      attrSet.add(AlarmRecordAttribute.LOGGING_TIME);
      attrSet.add(AlarmRecordAttribute.EVENT_TYPE);
      attrSet.add(AlarmRecordAttribute.ACK_STATE);
      attrSet.add(AlarmRecordAttribute.ACK_OPERATOR);
      attrSet.add(AlarmRecordAttribute.MANAGED_OBJECT_INSTANCE);
      attrSet.add(AlarmRecordAttribute.LOG_NAME);
      attrSet.add(AlarmRecordAttribute.MANAGED_OBJECT_INSTANCE);

      //Get all the alarms satisfying the query created above.
      System.out.println("AlarmLog query begins");
      AlarmRecord[] alarms = log.getAlarms(query, attrSet);

      //Now print out the result
      //------------------------
      printAlarmRecord(alarms, attrSet);

    }
    catch (Exception e) {
      e.printStackTrace();
      System.exit(-1);
    }

    System.out.println("Done.");
    System.exit(0);
  }

  private static void printAlarmRecord(AlarmRecord[] alarms,
```

CODE EXAMPLE A-1 GetAlarmsForNode.java *(Continued)*

```
                                    AlarmRecordAttributeSet attrSet)
    throws AlarmException
  {
   System.out.println("Received : " + alarms.length + " Alarms ");
    int ii;
    for (ii=0; ii<alarms.length; ii++)
      {
        System.out.println("Alarm number:" + ii );
        AlarmRecord alr1 = (AlarmRecord)alarms[ii];
        System.out.println(alr1.toString());
      }

  }

  private static final String sccsID =
        "@(#)GetAlarmsForNode.java       1.3 98/10/30 Sun
Microsystems";
}
```

Index

D

E

F

G

I

J

L

M

N

O

P

R

S

T

V