BEA AquaLogic Service Bus™

JMX Monitoring API Programming Guide

Version 2.5
Document Revised: October 2006

Contents: JMX Monitoring API

Introduction

Purpose of JMX Monitoring API

In previous releases of AquaLogic Service Bus, run-time service-monitoring data was only available through the Monitoring Dashboard of the AquaLogic Service Bus console. It was not possible for clients to programmatically access and consume monitoring data.

The JMX Monitoring API in AquaLogic Service Bus provides external access to monitoring data. Java Management Extensions (JMX) technology was used for the implementation, as it is a public standard that meets the implementation requirements.

The primary purpose of the JMX Monitoring API is to provide efficient, lower-level APIs supporting bulk operations. It does this using JMX as a transport. This API is not a high-level API compatible with JMX-based tools. However, if you are developing client software, you may want to develop high-level JMX APIs that support JMX-based tooling.

Description

The JMX monitoring API makes use of JMX as a transport only.

It exposes a public MBean to provide all the required operations to get monitoring data (statistical information) for any monitored service and its components.

It also exposes a set of public POJO objects required to carry out operations provided by the MBean.

There is no need for third-party client software to know the intricacies of the hierarchy inherent in the statistical information stored in the AquaLogic Service Bus monitoring system.

Using these APIs, customers can integrate their monitoring/management systems with AquaLogic Service Bus to do the following:

- Identify services enabled for monitoring.
- Get detailed statistical information for a specific service, for its components, or for both.
- Reset statistics accumulated since the last reset.

More Information

For more information about AquaLogic Service Bus 2.5, see BEA AquaLogic Service Bus 2.5.

For basic concepts and terminology, see AquaLogic Service Bus Concepts and Architecture on that page.

Concepts

The public JMX APIs are modeled by a single instance of `ServiceDomainMBean`, which has operations to check for monitored services and retrieve data from them.

A public set of POJOs provide additional objects and methods that, along with `ServiceDomainMbean`, provide a complete API for monitoring statistics.

The following sections provide brief descriptions of the POJOs and MBean. The Javadoc provides detailed descriptions. There is also a detailed description of statistics that are reported for resources.

Please be sure to read the important notes at the end of this chapter.

Public POJO Objects

The following POJO objects are exposed as part this API.

ResourceType

ServiceResourceStatistic

ResourceStatistic

StatisticValue

StatisticType

ResourceType

This object represents all types of resources that are enabled for service monitoring. There are three `enum` constants representing types: `SERVICE`, `FLOW_COMPONENT`, and `WEBSERVICE_OPERATION`.

See `com.bea.wli.monitoring.ResourceType` in the javadoc at:

`http://edocs.bea.com/alsb/docs25/javadoc/`

ServiceResourceStatistic

This object represents all business and proxy service resource types and the statistics associated with them. There are methods to get statistics for all resources or for a specified one.

See `com.bea.wli.monitoring.ServiceResourceStatistic` in the javadoc at:

`http://edocs.bea.com/alsb/docs25/javadoc/`

ResourceStatistic

This object represents a resource for which statistics collection is supported. There are methods to get the name of the resource, the type, and the statistics.

See `com.bea.wli.monitoring.ResourceStatistic` in the javadoc at:

`http://edocs.bea.com/alsb/docs25/javadoc/`

StatisticValue

This object represents a statistic value for a resource. The monitoring system currently supports the following types of statistic values, both nested classes:

- `CountStatistic`
- `IntervalStatistic`

`StatisticValue` is an abstract class so that concrete objects representing count and interval statistic values can be derived from it. It includes `getName()` and `getType()` methods.

See `com.bea.wli.monitoring.StatisticValue` in the javadoc at:

`http://edocs.bea.com/alsb/docs25/javadoc/`

StatisticType

This object represents predefined types of statistics. There are two `enum` types: `COUNT` and `INTERVAL`.

See `com.bea.wli.monitoring.StatisticValue` in the javadoc at:

`http://edocs.bea.com/alsb/docs25/javadoc/`

ServiceDomainMBean

This is the only MBean exposed as part of the public JMX API. It provides methods to find monitored service and get and reset statistics.

See `com.bea.wli.monitoring.ServiceDomainMBean` in the javadoc at:

`http://edocs.bea.com/alsb/docs25/javadoc/`

Statistics Details

The following sections provide detailed information about statistics reported for each resource type.

Statistics details for resource type - SERVICE

A service is an inbound or outbound endpoint that is configured within AquaLogic Service Bus. It may have an associated WSDL, security settings, etc.

The following statistics are reported for this resource type.

Table 2-1 SERVICE Statistics

Statistic Name	Type
message-count	count
error-count	count
failover-count	count
wss-error	count
response-time	interval

Table 2-1 SERVICE Statistics

Statistic Name	Type
validation-errors	count
failure-rate	count
success-rate	count
severity-warning	count
severity-major	count
severity-minor	count
severity-normal	count
severity-fatal	count
severity-critical	count
severity-all	count

Notes:

1. Statistic “wss-error” provides Web Service security violations counts. It is applicable to both business and proxy services.
2. Statistic “validation errors” is only applicable to proxy service; it is not returned for a business service.
3. Statistic “failover-count” is only applicable to business service; it is not returned for a proxy service.
4. All severity counts (“severity-warning”, “severity-major”, “severity-minor”, “severity-normal”, “severity-fatal”, “severity-critical”, and “severity-all”) give the number of generated alerts by severity type.

Statistics details for resource type – FLOW_COMPONENT

Statistics are collected for the following two types of components that can be present in the flow definition of a proxy service.

- Pipeline-pair node
- Route node

Pipelines are one-way processing paths consisting of stages that are executed sequentially against the current message. Stages are used to perform activities such as transformation, logging and publishing.

There are three categories of pipelines: request, response, and error.

The pipeline-pair node ties together a single request and a single response pipeline into one top-level element.

A **routing node** consists of a set of routes. A route identifies a target service and includes some additional configuration options that determines how the message will be packaged and sent to that service. A routing node will result in at most one route being selected as part of request processing.

The following statistics are reported for this resource type.

Table 2-2 FLOW_COMPONENT Statistics

Statistic Name	Type
elapse-time	interval
message-count	count
error-count	count

Notes:

1. Statistics for pipeline and route nodes are returned as statistics for flow components. `enum` value `ResourceType.FLOW_COMPONENT` represents both pipeline and route nodes.
2. Thus there is no way for a client to check if the returned flow component is a pipeline or route node. The name of the flow component, however, may suggest the type.

Statistics details for resource type – WEBSERVICE_OPERATION

This resource type provides statistical information pertaining to WSDL operations. Statistics are reported for each defined operation.

The following statistics are reported.

Table 2-3 WEBSERVICE_OPERATION Statistics

Statistic Name	Type
elapsed-time	interval
message-count	count
error-count	count

Caveats

Please be aware of the following:

- A client program will not know about newly added services that have monitoring turned on, or services modified to turn on monitoring, unless it periodically checks for such changes.
- After performing a reset operation on services, a client must wait at least a minute before retrieving statistics for those services so that statistics are updated by the reset operation. This is required, since, for performance reasons, this API is not designed to return real-time statistical data; rather, it uses a snapshot of statistical data that gets refreshed every minute. A statistics-collection timestamp can be used to determine if data is retrieved from the same snapshot or not.

  In order to ensure that data is retrieved from a snapshot taken after the reset operation, check if the statistics-collection timestamp is greater than the reset-request time.
- Because this API is based on snapshots of statistical data that gets refreshed every minute, and because reset is an expensive operation, reset operations should not be performed too frequently. BEA recommends that reset intervals be greater than 15 minutes.
- BEA strongly discourages using this API in a concurrent manner with more than one thread or process. This is because a reset performed in one thread or process is not visible to another threads or processes. This caveat also applies to resets performed from the Monitoring Dashboard of the AquaLogic Service Bus Console, as such resets are not visible to this API.

Performance

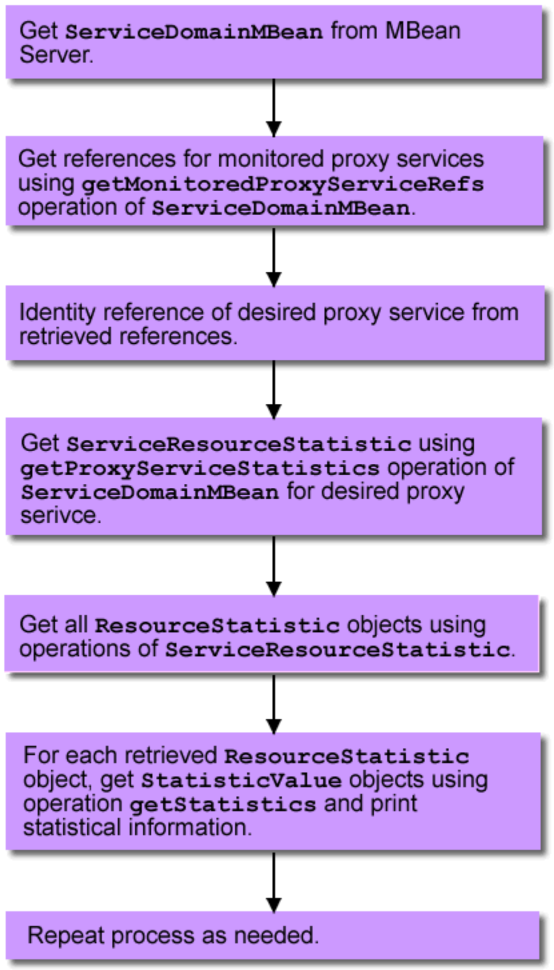
Performance should be better than or equivalent to that observed in the Monitoring Dashboard of the AquaLogic Service Bus Console.

API Usage Example

The sample program in this section demonstrates how to use the JMX Monitoring API.

The following figure depicts how statistics can be retrieved for a proxy service that is enabled for monitoring.

Figure 3-1 Retrieving Statistics for a Proxy


Get ServiceDomainMBean from MBean Server.
Get references for monitored proxy services using getMonitoredProxyServiceRefs operation of ServiceDomainMBean.
Identity reference of desired proxy service from retrieved references.
Get ServiceResourceStatistic using getProxyServiceStatistics operation of ServiceDomainMBean for desired proxy serivce.
Get all ResourceStatistic objects using operations of ServiceResourceStatistic.
For each retrieved ResourceStatistic object, get StatisticValue objects using operation getStatistics and print statistical information.
Repeat process as needed.

Sample Program

The following sample program explains how to:

1. Find by type services enabled for monitoring.
2. Set a resource-type filter.
3. Retrieve by type statistics for one or more services.
4. Extract statistics from the `ServiceResourceStatistics` object.
5. Save retrieved statistics in the proper format.
6. Reset statistics for one or more services.

To run this program, include the following JAR files in the classpath.

1. `weblogic.jar`
2. `sb-public.jar`

You may need to reset the default values for `HOSTNAME`, `PORT`, `USERNAME`, and/or `PASSWORD` in the code below for your environment.

For performance reasons, avoid extracting and resetting statistics for a large number of services too often. See "Caveats" on page 2-6.

```
package tests;

import com.bea.wli.config.Ref;
import com.bea.wli.monitoring.*;
import weblogic.management.jmx.MBeanServerInvocationHandler;

import javax.management.MBeanServerConnection;
import javax.management.MalformedObjectNameException;
import javax.management.ObjectName;
import javax.management.remote.JMXConnector;
import javax.management.remote.JMXConnectorFactory;
import javax.management.remote.JMXServiceURL;
import javax.naming.Context;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;
import java.net.MalformedURLException;
import java.util.*;
import java.text.SimpleDateFormat;

/**
 * This class provides sample code showing how to use
 * ServiceDomainMBean.
 * It shows how to:
 *  1. Find business and proxy services enabled for monitoring.
 *  2. Get statistics for one or more business and proxy services.
 *  3. Perform reset operation on one or more business and proxy services.
 *  4. Handle exceptions.
 * It uses a timer to retrieve statistics, save them in a file, and
 * perform resets in a recursive manner.
 */
public class ServiceStatisticsRetriever {

   private ServiceDomainMBean serviceDomainMbean = null;

   /**
    * Retrieve statistics for all business services being monitored in the
    * domain and reset statistics for the same.
    * @throws Exception
    */
   void getAndResetStatsForAllMonitoredBizServices() throws Exception {
      Ref[] serviceRefs =
         serviceDomainMbean.getMonitoredBusinessServiceRefs();

      // Create a bitwise map for desired resource types.
      int typeFlag = 0;
      typeFlag = typeFlag | ResourceType.SERVICE.value();
      typeFlag = typeFlag | ResourceType.WEBSERVICE_OPERATION.value();

      HashMap<Ref, ServiceResourceStatistic> resourcesMap = null;
      // Get cluster-level statistics.
      try {
         // Get statistics.
         System.out.println("Now trying to get statistics for -" +
            serviceRefs.length + " business services...");
         resourcesMap =
            serviceDomainMbean.getBusinessServiceStatistics(serviceRefs,
            typeFlag, null);

         // Reset statistics.
         long resetRequestTime =
            serviceDomainMbean.resetStatistics(serviceRefs);

         // Save retrieved statistics.
         String fileName = "BizStatistics_" +
            new SimpleDateFormat("yyyy_MM_dd_HH_mm").
               format(new Date(System.currentTimeMillis())) + ".txt";
         saveStatisticsToFile(resourcesMap, resetRequestTime, fileName);
      }
      catch (IllegalArgumentException iae) {
         System.out.println("===================================\n");
         System.out.println("Encountered IllegalArgumentException...
            Details:");
         System.out.println(iae.getMessage());
         System.out.println("Check if proxy ref was passed OR flowComp " +
            "resource was passed OR bitmap is invalid..." +
            "\nIf so correct it and try again!!!");
         System.out.println("===================================\n");
         throw iae;
      }
      catch (DomainMonitoringDisabledException dmde) {
         /** Statistics not available as monitoring is turned off at domain
          * level.
          */
         System.out.println("===================================\n");
         System.out.println("Statistics not available as monitoring " +
            "is turned off at domain level.");
         System.out.println("===================================\n");
         throw dmde;
      }
      catch (MonitoringException me) {
         // Internal problem... May be aggregation server is crashed...
         System.out.println("===================================\n");
         System.out.println("ERROR: Statistics is not available... " +
            "Check if aggregation server is crashed...");
         System.out.println("===================================\n");
         throw me;
      }
   }
```

```
/**
 * Retrieve statistics for all proxy services being monitored in the
 * domain and reset statistics for the same.
 * @throws Exception
 */
void getAndResetStatsForAllMonitoredProxyServices() throws Exception {
   Ref[] serviceRefs =
      serviceDomainMbean.getMonitoredProxyServiceRefs();

   // Create a bitwise map for desired resource types.
   int typeFlag = 0;
   typeFlag = typeFlag | ResourceType.SERVICE.value();
   typeFlag = typeFlag | ResourceType.FLOW_COMPONENT.value();
   typeFlag = typeFlag | ResourceType.WEBSERVICE_OPERATION.value();

   HashMap<Ref, ServiceResourceStatistic> resourcesMap = null;

   // Get cluster-level statistics.
   try {
      // Get statistics.
      System.out.println("Now trying to get statistics for -" +
         serviceRefs.length + " proxy services...");
      resourcesMap =
         serviceDomainMbean.getProxyServiceStatistics(serviceRefs,
            typeFlag, null);

      // Reset statistics.
      long resetRequestTime =
         serviceDomainMbean.resetStatistics(serviceRefs);

      // Save retrieved statistics.
      String fileName = "ProxyStatistics_" +
         new SimpleDateFormat("yyyy_MM_dd_HH_mm").
            format(new Date(System.currentTimeMillis())) + ".txt";
      saveStatisticsToFile(resourcesMap, resetRequestTime, fileName);
   }
   catch (IllegalArgumentException iae) {
      System.out.println("===================================\n");
      System.out.println("Encountered IllegalArgumentException...
         Details:");
      System.out.println(iae.getMessage());
      System.out.println("Check if business ref was passed OR bitmap
         is " + "invalid...\nIf so correct it and try again!!!");
      System.out.println("===================================\n");
      throw iae;
   }
   catch (DomainMonitoringDisabledException dmde) {
      /** Statistics not available as monitoring is turned off at the
       * domain level.
       */
      System.out.println("===================================\n");
      System.out.println("Statistics not available as monitoring " +
         "is turned off at domain level.");
      System.out.println("===================================\n");
      throw dmde;
   }
   catch (MonitoringException me) {
      // Internal problem ... May be aggregation server is crashed ...
      System.out.println("===================================\n");
      System.out.println("ERROR: Statistics is not available... " +
         "Check if aggregation server is crashed...");
      System.out.println("===================================\n");
      throw me;
   }
}

/**
 * Saves statistics of all services from the specified map.
 * @param statsMap Map containing statistics for one or more services
 * of the same type; i.e., business or proxy.
 * @param resetReqTime Reset request time. This information will be
 * written at the end of the file, provided it is not zero.
 * @param fileName Statistics will be saved in a file with this name.
 * @throws Exception
 */
private void saveStatisticsToFile(
      HashMap<Ref, ServiceResourceStatistic> statsMap,
         long resetReqTime, String fileName) throws Exception {
   if (statsMap == null) {
      System.out.println("\nService statistics map is null... Nothing to
      save.\n");
   }
   if (statsMap.size() == 0) {
      System.out.println("\nService statistics map is empty... Nothing
                to save.\n");
            }
            FileWriter out = new FileWriter(new File(fileName));

            out.write("*****************************************************\n";
            out.write("This file contains statistics for " + statsMap.size() +
               " services.\n");
            out.write("*****************************************************\n")
      ;

            Set<Map.Entry<Ref, ServiceResourceStatistic>> set =
               statsMap.entrySet();

            System.out.println(new StringBuffer().append("\nWriting stats to the
               file - ").append(fileName).append("\n").toString());

            // Print statistical information of each service
            for (Map.Entry<Ref, ServiceResourceStatistic> mapEntry : set) {
               out.write(new StringBuffer().
                  append("\n\n======= Pirnting statistics for service ").
                  append(mapEntry.getKey().getFullName()).
                  append("=======\n").toString());

               ServiceResourceStatistic serviceStats = mapEntry.getValue();
               out.write(new StringBuffer().
                  append("Statistic collection time is - ").
                  append(new Date(serviceStats.getCollectionTimestamp())).
                  append("\n").toString());


               ResourceStatistic[] resStatsArray = null;
               try {
                  resStatsArray = serviceStats.getAllResourceStatistics();
               }
               catch (MonitoringNotEnabledException mnee) {
                  // Statistics not available
                  out.write("WARNING: Monitoring is not enabled for " +
                     "this service... Do someting...");
                  out.write("=================================================\n";
                  continue;
               }

               catch (InvalidServiceRefException isre) {
                  // Invalid service
```

```
    out.write("ERROR: Invlaid Ref. May be this service is " +
        "deleted. Do something...");
    out.write(
        "===============================================\n");
    continue;
}
catch (MonitoringException me) {
    // Statistics not available
    out.write("ERROR: Failed to get statistics for this
        service... " + "Details: " + me.getMessage());
    me.printStackTrace();
    out.write(
        "===============================================\n");
    continue;
}

for (ResourceStatistic resStats : resStatsArray) {
    // Print resource information
    out.write("\nResource name: " + resStats.getName());
    out.write("\tResource type: " +
        resStats.getResourceType().toString());

    // Now get and print statistics for this resource
    StatisticValue[] statValues = resStats.getStatistics();
    for (StatisticValue value : statValues) {
        out.write("\n\t\tStatistic Name - " + value.getName());
        out.write("\n\t\tStatistic Type - " +
            value.getType().toString());

        // Determine statistics type
        if ( value.getType() == StatisticType.INTERVAL ) {
            StatisticValue.IntervalStatistic is =
                (StatisticValue.IntervalStatistic)value;

            // Print interval statistics values
            out.write("\n\t\t\t\tCount Value - " + is.getCount());
            out.write("\n\t\t\t\tMin Value - " + is.getMin());
            out.write("\n\t\t\t\tMax Value - " + is.getMax());
            out.write("\n\t\t\t\tSum Value - " + is.getSum());
            out.write("\n\t\t\t\tAve Value - " + is.getAverage());
        }
        else if ( value.getType() == StatisticType.COUNT ) {
                StatisticValue.CountStatistic cs =
                   (StatisticValue.CountStatistic)value;

                // Print count statistics value
                out.write("\n\t\t\t\tCount Value - " + cs.getCount());
             }
          }
       }
       out.write(
          "\n===============================================\n");
    }
    if (resetReqTime > 0) {
       // Save reset request time.
       out.write(
          "\n***************************************************\n");
       out.write("Statistics for all these services are RESET.\n");
       out.write("RESET request time is " +
          new SimpleDateFormat("MM/dd/yyyy HH:mm:ss").
             format(new Date(resetReqTime)));
       out.write(
          "\n***************************************************\n");
    }

    // Flush and close file.
    out.flush();
    out.close();
 }

 /**
  * Init method.
  *
  * @param props Properties required for initialization.
  * @throws Exception
  */
 private void init(HashMap props) throws Exception {
    Properties properties = new Properties();
    properties.putAll(props);
    getServiceDomainMBean(properties.getProperty("HOSTNAME"),
       Integer.parseInt(properties.getProperty("PORT", "7001")),
       properties.getProperty("USERNAME"),
       properties.getProperty("PASSWORD"));
 }
```

```
/**
 * Gets an instance of ServiceDomainMBean from the weblogic server.
 *
 * @param host
 * @param port
 * @param username
 * @param password
 * @throws Exception
 */
private void getServiceDomainMBean(String host, int port, String
      username, String password) throws Exception {
   InvocationHandler handler =
      new ServiceDomainMBeanInvocationHandler(host, port, username,
         password);
   Object proxy = Proxy.newProxyInstance(
      ServiceDomainMBean.class.getClassLoader(),
      new Class[]{ServiceDomainMBean.class}, handler);
   serviceDomainMbean = (ServiceDomainMBean) proxy;
}

/**
 * Invocation handler class for ServiceDomainMBean class.
 */
public static class ServiceDomainMBeanInvocationHandler
      implements InvocationHandler {
   private String jndiURL =
      "weblogic.management.mbeanservers.domainruntime";
   private String mbeanName = ServiceDomainMBean.NAME;
   private String type = ServiceDomainMBean.TYPE;


   private String protocol = "t3";
   private String hostname = "localhost";
   private int port = 7001;
   private String jndiRoot = "/jndi/";

   private String username = "weblogic";
   private String password = "weblogic";

   private JMXConnector conn = null;
   private Object actualMBean = null;
```

```
public ServiceDomainMBeanInvocationHandler(String hostName, int port,
      String userName, String password) {
   this.hostname = hostName;
   this.port = port;
   this.username = userName;
   this.password = password;
}

/**
 * Gets JMX connection
 * @return JMX connection
 * @throws IOException
 * @throws MalformedURLException
 */
public JMXConnector initConnection()
      throws IOException, MalformedURLException {
   JMXServiceURL serviceURL = new JMXServiceURL(protocol,
      hostname, port, jndiRoot + jndiURL);
   Hashtable<String, String> h = new Hashtable<String, String>();

   if (username != null)
      h.put(Context.SECURITY_PRINCIPAL, username);
   if (password != null)
      h.put(Context.SECURITY_CREDENTIALS, password);

   h.put(JMXConnectorFactory.PROTOCOL_PROVIDER_PACKAGES,
      "weblogic.management.remote");
   return JMXConnectorFactory.connect(serviceURL, h);
}

/**
 * Invokes specified method with specified params on specified
 * object.
 * @param proxy
 * @param method
 * @param args
 * @return
 * @throws Throwable
 */
public Object invoke(Object proxy, Method method, Object[] args)
      throws Throwable {
   try {
      if (conn == null) {
         conn = initConnection();
        }
      if (actualMBean == null) {
            actualMBean =
               findServiceDomain(conn.getMBeanServerConnection(),
               mbeanName, type, null);
         }

         Object returnValue = method.invoke(actualMBean, args);


         return returnValue;
      }
      catch (Exception e) {
         throw e;
      }
   }

   /**
    * Finds the specified MBean object
    *
    * @param connection - A connection to the MBeanServer.
    * @param mbeanName - The name of the MBean instance.
    * @param mbeanType - The type of the MBean.
    * @param parent - The name of the parent Service. Can be NULL.
    * @return Object - The MBean or null if the MBean was not found.
    */
   public Object findServiceDomain(MBeanServerConnection connection,
         String mbeanName,
         String mbeanType,
         String parent) {
      ServiceDomainMBean serviceDomainbean = null;
      try {
         ObjectName on =
            newObjectName(ServiceDomainMBean.OBJECT_NAME);
         serviceDomainbean = (ServiceDomainMBean)
            MBeanServerInvocationHandler.
               newProxyInstance(connection, on);
      }
      catch (MalformedObjectNameException e) {
         e.printStackTrace();
         return null;
      }
      return serviceDomainbean;
   }
}
```

```
/**
 * Timer task to keep retrieving and resetting service statistics.
 */
static class GetAndResetStatisticsTask extends TimerTask {
   private ServiceStatisticsRetriever collector;

   public GetAndResetStatisticsTask(ServiceStatisticsRetriever col){
      collector = col;
   }

   public void run() {
      System.out.println("\n**************************************");
      System.out.println("Retrieving statistics for all monitored " +
         "business services.");
      try {
         collector.getAndResetStatsForAllMonitoredBizServices();
         System.out.println("Successfully retrieved and reset
            statistics for " +
            "all monitored \n business services at " +
            new SimpleDateFormat("MM/dd/yyyy HH:mm:ss").
               format(new Date(System.currentTimeMillis())));
      } catch (Exception e) {
         System.out.println("Failed to retrieve and reset statistics
            for all " + "monitored business service...");
         e.printStackTrace();
      }
      System.out.println("**************************************\n");

      System.out.println("\n**************************************");
      System.out.println("Retrieving statistics for all monitored
         proxy services.");
      try {
         collector.getAndResetStatsForAllMonitoredProxyServices();
         System.out.println("Successfully retrieved and reset
            statistics " +
            "for all monitored \nproxy services at " +
            new SimpleDateFormat("MM/dd/yyyy HH:mm:ss").
               format(new Date(System.currentTimeMillis())));
      } catch (Exception e) {
            System.out.println("Failed to retrieve and reset
               statistics " + "for all monitored proxy service...");
            e.printStackTrace();
      }
      System.out.println("***********************************\n");
   }
}

/**
 * The main method to start the timer task to extract, save, and reset
 * statistics for all monitored business and proxy services. It uses
 * the following system properties.
 * 1. hostname - Hostname of admin server
 * 2. port - Listening port of admin server
 * 3. username - Login username
 * 4. password - Login password
 * 5. period - Frequency in hours. This will be used by the timer
 * to determine the time gap between two executions.
 *
 * @param args Not used.
 */
public static void main(String[] args) {
   try {
         Properties p = System.getProperties();

         HashMap map = new HashMap();

         map.put("HOSTNAME", p.getProperty("hostname",
               "localhost"));
         map.put("PORT", p.getProperty("port", "7001"));
         map.put("USERNAME", p.getProperty("username", "weblogic"));
         map.put("PASSWORD", p.getProperty("password", "weblogic"));

         ServiceStatisticsRetriever collector =
            new ServiceStatisticsRetriever();
         String periodStr = p.getProperty("period", "1");
         int periodInHour = Integer.parseInt(periodStr);
         long periodInMilliSec = periodInHour * 60 * 60 * 1000;

         collector.init(map);

         // Start timer.
         Timer timer = new Timer();
         timer.scheduleAtFixedRate(
            new GetAndResetStatisticsTask(collector),
                  0, periodInMilliSec);
         }
         catch (Exception e) {
            e.printStackTrace();
         }
      }
   }
```