WebSphere™ Application Server

Using the Performance Monitoring Infrastructure Client Package

Version 4.0

GC09-3951-00

Contents

Figures

Tables

Using the Performance Monitoring Infrastructure Client Package

Introduction

The Performance Monitoring Infrastructure (PMI) is a set of packages and libraries designed to assist with gathering, delivering, processing, and displaying performance data in WebSphere Application Server Advanced Edition domains. This document discusses the client packages of the PMI application programming interface (API) and describes how to use them to write WebSphere Application Server clients that collect and display performance data from servers.

PMI organization and implementation

PMI follows a client/server architecture. In PMI terms, a server is any application that uses the PMI API to collect performance data; servers can include application servers, HTTP servers, and Java applications. WebSphere Application Server provides a server named PerfServer that is responsible for retrieving performance data from other servers in the domain and making the data available to interested clients, as shown in Figure 1 on page 2. A client is an application that receives performance data from a server or servers and processes the data; clients can include graphical user interfaces (GUIs) that display performance data in real time, applications that monitor performance data and trigger different events according to the current values of the data, or any other application that needs to receive and process performance data.

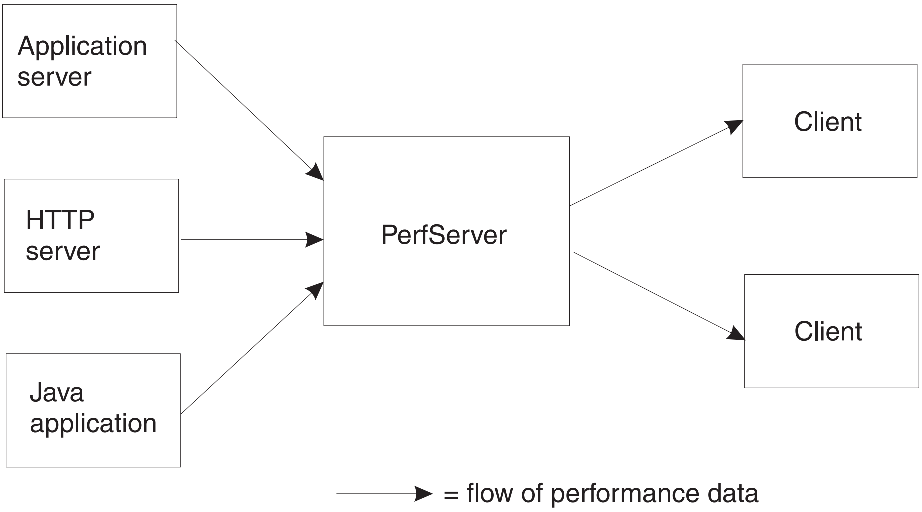

Figure 1. The role of PerfServer in collecting and distributing performance data

Each piece of performance data has two components, a static component and a dynamic component. The static component consists of a name and an ID to identify the data, as well as other descriptive attributes that assist the client in processing and displaying the data. The dynamic component consists of information that changes over time, such as the current value of a counter and the time stamp associated with that value.

Performance data is classified into the following four types:

- *Numeric data*, consisting of a single numeric value such as an integer, a long, or a double. It is used to represent data such as counts and sizes.
- *Statistical data* on a sample space. It consists of the number of elements in the sample set, the sum of the elements, and the sum of squares. These values can be used to obtain the mean, the variance, and the standard deviation of the mean. An example of statistical data is the response time for each invocation of an enterprise bean.
- *Load data*, which monitors a value as a function of time. Example uses include tracking the number of threads or the number of service requests in a queue. Load data tracks the current value, the time the value was reached, and the integral over time of the value. These values can be used to obtain the weighted average for the level over a period of time. An example of load data is the average size of a database connection pool during a specified time interval.
- *Group data* is a collection of performance data intended to be used by groups. It enables servers to create sets of performance data that can be retrieved by clients with a single call.

Data is organized by modules; each module has a configuration file in extensible markup language (XML) format that determines its organization. The configuration file lists a unique identifier for each piece of performance data in the module. A client can use the data's unique ID to retrieve the data's static information; the server then sends the dynamic information associated with that data to the client. A server can track many instances of each type of performance data—for example, a number of pieces of performance data tracking the average response time of bean methods. In this case, each piece of performance data shares the same ID, and the server sends additional identifying information (for example, the bean's home name) along with the performance data so that clients can distinguish among the different instances.

PMI interfaces with WebSphere administration utilities to enable administrators to control the amount and level of performance data collected. You can access the PMI administrative interface by using the Administrative Console.

PMI client interfaces

This section discusses PMI's client implementation, including the organization of data sent to clients and the interfaces clients use to retrieve and process performance data from servers. Performance data used by PMI's client implementation is referred to as *client performance data* (CPD).

Data organization and hierarchy

PMI data is provided to clients in a hierarchical structure. The CpdSnapshot object is the root of the hierarchy. Descending from the CpdSnapshot object are node information, server information, module information, and PerfCollection and CpdData objects. See Figure 2 on page 4 for a diagram of the data hierarchy. Note that the node-information and server-information objects contain no performance data.

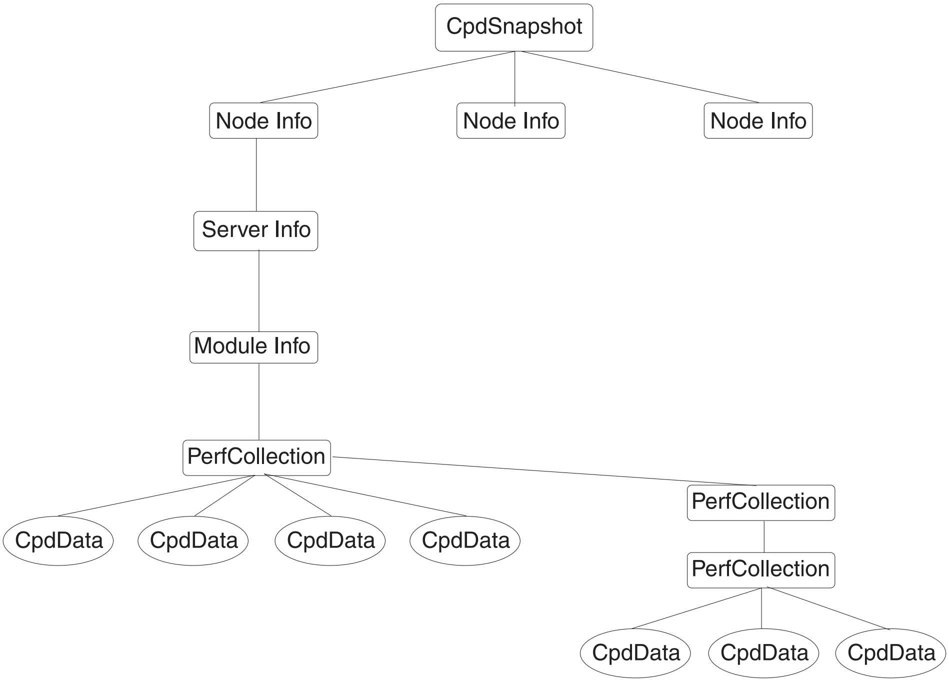

Figure 2. Organization of PMI data

Each time a client retrieves performance data from a server, the data is returned in a subset of this structure; the form of the subset depends on the data that is retrieved. You can update the entire structure with new data or update only part of the tree, as needed.

PMI interfaces

The PMI PerfServer exports the CpdCollection, CpdData, and CpdValue interfaces to provide performance data to interested clients. The PMI API provides the PmiClient interface to enable clients to receive performance data from servers. For details on these interfaces, see “The CpdCollection interface”, “The CpdData and CpdValue objects” on page 7, and “The PmiClient class” on page 9. In addition, PMI provides the CpdEventListener and CpdEvent interfaces to enable clients to register as listeners, and thus to be informed when new or changed data is available at the server; see “The CpdEventListener and CpdEvent interfaces” on page 11 for details. Finally, PMI provides the CpdFamily class to assist with displaying data in table form; see “The CpdFamily class” on page 12 for details.

The CpdCollection interface

The CpdCollection interface is the base interface to PMI. It organizes performance data in the hierarchy described in “Data organization and hierarchy" on page 3. Each member of the hierarchy is an instance of CpdCollection that contains a number of data members and a number of CpdCollection children.

The CpdCollection interface extends two other PMI interfaces, CpdXML and CpdEventSender. These interfaces are defined as follows:

```
public interface CpdCollection extends Serializable, CpdXML,
  CpdEventSender {
     public PerfDescriptor getPerfDescriptor();
     public String getDescription();
     public int numDataMembers();
     public CpdData[] dataMembers();
     public CpdData getData(int index);
     public int numSubcollection();
     public CpdCollection[] subcollections();
     public CpdCollection getSubcollection(int i);
     public CpdCollection findCollection(PerfDescriptor pd);
     public void addSubcollection(CpdCollection col);
     public CpdCollection getParent();
     public void update(CpdCollection other);
     public CpdCollection reset();
}

public interface CpdXML {
     public String toXML();
     public void fromXML(String xmlStr);
}

public interface CpdEventSender extends Cloneable {
     public void addCpdEventListener(CpdEventListener al);
     public void removeCpdEventListener(CpdEventListener al);
     public void notifyListeners(CpdEvent evt);
     public void notifyListeners(int evt_type);
}
```

Figure 3. Definitions of the CpdCollection, CpdXML, and CpdEventSender interfaces

The update method updates collections of data. To illustrate the functionality of this method, assume that the `collection1.update(collection2)` statement is used to update a data collection named `collection1` with the data in a collection named `collection2`. In this case, the update method works as follows:

- If `collection1` and `collection2` represent the same collection (that is, if they are instances of the same PerfDescriptor object, with `collection2` representing a more recent version of the PerfDescriptor object than `collection1`), the update method performs the following tasks:
  - If any member of `collection2` does not have a corresponding member in `collection1`, the update method creates a child collection of `collection1` that contains the member from `collection2`.

  - For each member of `collection2` that has a corresponding member in `collection1`, the update method updates the member in `collection1` with the corresponding data in `collection2`.

  The update method then returns a value of `true` to the caller.
- If `collection1` and `collection2` do not represent the same collection, the update method performs the following tasks:
  - For any member of `collection1` that has a corresponding member in `collection2`, the update method updates the member in `collection1` with the corresponding data in `collection2`.
  - If `collection2` is a descendant of `collection1`, the update method creates a child collection of `collection1` and updates each member of the child collection with the corresponding data in `collection2`.

  If neither of these conditions is met, the update method returns a value of `false`.

The PerfDescriptor interface is used to specify the data that the client is interested in. It includes methods that return node name, server name, module name, collection name, and full name. Its definition is as follows:

```
public interface PerfDescriptor extends Serializable {
     public int getType(); // Types include node, server, module, instance,
          // and data
     public String getNodeName();
     public String getServerName();
     public String getModuleName();
     public String getName(); // Returns node, server, module, instance,
          // or data name, depending on type
     public String getFullName(); // Returns a name in the following form:
          // node.server.module.instance.data
     public String[] getPath();
     public boolean equals(PerfDescriptor pd);
     public boolean isDescendingFrom(PerfDescriptor pd);
     public int[] getDataIds(); // Returns all data IDs (null, one, or multiple)
          // in the descriptor
}
```

Figure 4. Definition of the PerfDescriptor interface

The PerfDescriptorList class is used to gather data from multiple PerfDescriptor instances. It includes methods to add, remove, and get PerfDescriptor instances. Its definition is as follows:

```
public class PerfDescriptorList {
     public boolean addDescriptor(PerfDescriptor pd); // If pd is not in the
          // list, add it and return true; otherwise, return false
     public boolean removeDescriptor(PerfDescriptor pd); // If pd is in the
          // list, remove it and return true; otherwise, return false
     public int numDescriptors(); // Return the number of PerfDescriptor
          // instances in the list
     public PerfDescriptor[] getDescriptors(); // Return all PerfDescriptors
          // in an array
}
```

Figure 5. Definition of the PerfDescriptorList interface

The CpdData and CpdValue objects

The CpdData object is the lowest level in the CPD hierarchy. Each CpdData instance contains all the static information for the performance data as well as a getValue method to return the data's dynamic information in the form of an instance of the CpdValue object. The CpdData interface provides an update method to take a reference to a new version of a piece of data and update the current object with the new value. The value is updated only if the new data has the same name as the original object. The CpdData interface also includes an addListener interface to enable data objects to register as event listeners; see "The CpdEventListener and CpdEvent interfaces" on page 11 for details. The CpdData interface extends the CpdXML and CpdEventSender interfaces, which are shown in Figure 3 on page 5.

The definition of CpdData is as follows:

```
public interface CpdData extends Serializable, CpdXML, CpdEventSender {
     public PerfDescriptor getDescriptor();
     public String getDescription();
     public void setValue(CpdValue value);
     public void update(CpdData other);
     public CpdValue getValue();
     public Object getParent();
     public void setParent(Object parent);
     public boolean reset();
}
```

Figure 6. Definition of the CpdData interface

A variety of data types extend the CpdValue interface. The interface provides the getValue, getTime, delta, and rate methods to work with data values. The definition of CpdValue is as follows:

```
public inteface CpdValue extends Serializable, Cloneable {
     public int getType();
     public long getTime();
     public double getValue();
     public CpdValue delta(CpdValue prev); // return the difference
     public CpdValue rate(CpdValue prev); // return the rate of the difference
     public void combine(CpdValue other); // add another value to this value
     public Object clone();
}
```

Figure 7. Definition of the CpdValue interface

Each client value type extends the CpdValue interface. The specific types are listed in Table 1.

Table 1. CpdValue types and associated methods

Type	Method	Description
CpdInt	int intValue()	Value as an int
CpdLong	long longValue()	Value as a long
CpdDouble	double doubleValue()	Value as a double
CpdStatData	double mean()	Mean of the sample set
	int count()	Element count
	double sumsquares()	Sum of squares of the elements
	double variance()	Variance
	double standardDeviation()	Standard deviation
	double confidence(int level)	Confidence interval of the mean
CpdLoad	double mean()	Time-weighted average value
	double getCurrentValue()	Last data point
	long getWeight()	Measured time period

The getValue method retrieves the value and, if possible, converts it to a double value. If it cannot make the conversion, it returns `Double.NaN`. The values returned by getValue can be used for displaying and graphing data.

The getTime method returns the server time associated with the data.

The delta method takes the current value and a previous value of a piece of data, and returns an object that represents the change between the values. The delta method also returns a deltaTime value, which represents the time associated with the delta value and the current value of the data. The delta method is defined for all objects listed in Table 1 on page 8. For CpdStatData, the delta between two values provides the statistics on all members of the current sample set, not on members of any previous set. The delta method is also defined for groups. For two groups, `g1` and `g2`, the object returned by the statement `g1.delta(g2)` is a group whose members include all members common to both `g1` and `g2`. For each member `m1` of group `g1` with a corresponding value of `m2` in `g2`, the corresponding delta value is represented by `m1.delta(m2)`.

The rate method returns the rate of change. This method is defined for the CpdInt, CpdLong, and CpdDouble types. If the rate cannot be calculated (for instance, if the method is used with the CpdStatData or CpdLoad types), the original value is returned.

For the CpdLoad object, the mean method returns the time-weighted average of the value being tracked. It is computed by dividing the integral value by the delta time. If the delta time is 0 (zero), the difference between the object's current time and its creation time is used.

The PmiClient class

The PmiClient class is used by clients to access performance data. It looks up session beans and invokes remote APIs, thus freeing the programmer from having to implement these tasks manually. A client can create an instance of PmiClient and call all subsequent methods on that object. The PmiClient object converts wire-level data to a client-side data collection hierarchy and exports methods for clients to create PerfDescriptor objects if the objects' names are known. If you know the static names for the node, server, module, instance, or data, you can call pmiClient.createPerfDescriptor to obtain the PerfDescriptor. Otherwise, you can get the names by issuing the listNodes, listServers, and listMembers methods on PmiClient.

The definition of PmiClient is as follows:

```
public class PmiClient {
     // Constructor: Look up a PerfRetrieve session bean home and
     //    create a bean object. Do all initialization (for example,
     //    get all configuration files).
     //    Default hostName is localhost; default port is 900
     //    Default JNDI name for perfRetrieveHome is "PerfRetrieveHome"
     PmiClient();
     PmiClient(String hostName);
     PmiClient(String hostName, String port);
     PmiClient(String hostName, String port, String perfRetrieveHome);

     // The top-level collection of the data hierarchy.
     CpdCollection createRootCollection();

     // The following methods serve as wrappers for the remote
     // methods in PerfRetrieve so that users do not need to
     // deal with remote APIs or wire-level data.

     // List all nodes in the domain, then call
     // PerfDescriptorInstance.getName() to get the node names.
     PerfDescriptor[] listNodes();

     // List all servers in a node; pd is the one returned from
     // listNodes. Call PerfDescriptorInstance.getName() to get
     // the server names.
     PerfDescriptor[] listServers(String nodeName);
     PerfDescriptor[] listServers(PerfDescriptor pd);

     // List the members in a server. The returned PerfDescriptor
     // can be passed to the next listMembers call until it
     // returns null (that is, when the leaf node is reached).
     PerfDescriptor[] listMembers(PerfDescriptor pd);

     // Get module configuration, which contains all the static
     // information for the data.
     PmiModuleConfig[] getConfigs();
     PmiModuleConfig[] getConfigs(String nodeName);
     PmiModuleConfig getConfig(String moduleID);

     // Retrieve performance data. The following modes are available:
     // - Single pd versus an array of pds
     // - With or without time interval
     // - Recursive versus nonrecursive (recursive retrieves data
     //   for each subgroup instead of aggregate data)
     CpdCollection get(PerfDescriptor pd, boolean recursive);
     CpdCollection get(PerfDescriptor pd, boolean recursive, int time);
     CpdCollection[] gets(PerfDescriptorList pds, boolean recursive);
     CpdCollection[] gets(PerfDescriptorList pds, boolean recursive,
          int time);
```

Figure 8. Definition of the PmiClient class (Part 1 of 2)

```
     // Retrieve performance data in XML format
     String getXML(PerfDescriptor pd, boolean recursive);
     String getXML(PerfDescriptor pd, boolean recursive, int time);
     String getXML(PerfDescriptorList pds, boolean recursive);
     String getXML(PerfDescriptorList pds, boolean recursive,
          int time);

     // Convert data ID and name
     public static String getDataName(String moduleID, int dataId);
     public static int getDataId(String moduleID, String name);

     // Methods to create a PerfDescriptor, used when you know
     // static names
     public PerfDescriptor createPerfDescriptor(){
     public PerfDescriptor createPerfDescriptor(String[] dataPath);
     public PerfDescriptor createPerfDescriptor(String[] dataPath,
          int dataId);
     public PerfDescriptor createPerfDescriptor(String[] dataPath,
          int[] dataIds);
     public PerfDescriptor createPerfDescriptor(PerfDescriptor parent,
          String name);
     public PerfDescriptor createPerfDescriptor(PerfDescriptor parent,
          int dataId);
     public PerfDescriptor createPerfDescriptor(PerfDescriptor parent,
          int[] dataIds);
     }
}
```

Figure 8. Definition of the PmiClient class (Part 2 of 2)

The CpdEventListener and CpdEvent interfaces

The PMI client package provides event and listener interfaces to inform clients (for instance, a GUI display) when new or changed data is available. The CpdEventObject interface, which extends java.util.EventObject, is the parent to the PMI event and listener interfaces. The CpdEventListener interface, which extends CpdEventObject, is the interface that objects need to implement to receive performance data events. Objects can use the addListener method to register as event listeners. The definition of the method is as follows:

```
void addListener(CpdEventListener listener);
```

The definitions of the CpdEventListener and CpdEvent interfaces are as follows:

```
public interface CpdEventListener {
     public void CpdEventPerformed(CpdEvent evt);
}

public class CpdEvent {
     final static int EVENT_NEW_MEMBER = 0;
     final static int EVENT_NEW_SUBCOLLECTION = 1;
     final static int EVENT_NEW_DATA = 2;

     private int type;
     private Object source = null;

     public CpdEvent(Object source, int type);
     public CpdEvent(int type);
     public Object getSource();
     public int getType();
}
```

Figure 9. Definitions of the CpdEventListener and CpdEvent interfaces

The CpdFamily class

The PMI client provides the CpdFamily class to simplify displaying data in a table. When two data objects have the same module identifier, they are in the same family and can be displayed in the same table by using this class. The definition of CpdFamily is as follows:

```
public class CpdFamily {
     static public boolean isSameFamily(CpdData d1, CpdData d2);
     static public boolean isSameRow(CpdData d1, CpdData d2);
     static public boolean isSameColumn(CpdData d1, CpdData d2);
     static public boolean getRow(CpdData d1);
     static public boolean getColumn(CpdData d1);
     static public boolean getFamilyName(CpdData d1);
}
```

Figure 10. Definition of the CpdFamily class

Using the PMI client interfaces

This section discusses the use of the PMI client interfaces in applications. The basic programming model is as follows:

1. A client uses the CpdCollection interface to retrieve an initial collection, or snapshot, of performance data from the server. This snapshot, which is called `Snapshot` in this example, is provided in a hierarchical structure as described in “Data organization and hierarchy” on page 3, and contains the current values of all performance data collected by the server. The snapshot maintains the same structure throughout the lifetime of the CpdCollection instance.

2. The client processes and displays the data as specified. Processing and display objects (for example, filters and GUIs) can register as CpdEvent listeners to data of interest; see “The CpdEventListener and CpdEvent interfaces” on page 11 for details. When the client receives updated data, all listeners are notified.
3. When the client collects new or changed data (for example, data collections named `S1`, `S2`, and so on) from the server, the client uses the update method to update `Snapshot` with the new data:

   ```
   Snapshot.update(S1);
   // ...later...
   Snapshot.update(S2);
   ```
4. Step 2 and Step 3 are repeated through the lifetime of the client.

Figure 11 lists a sample of PMI client code.

```
import com.ibm.websphere.pmi.*;
import com.ibm.websphere.pmi.server.*;
import com.ibm.websphere.pmi.client.*;

public class PmiTest implements PmiConstants {

   // A test driver
   // If arguments are provided:
   //   args[0] = node name
   //   args[1] = port number
   //   args[2] = The JNDI name of PerfRetrieve
   //
   // Note: This will not work unless an admin server and
   // perfServer are running
   //

public static void main(String[] args) {
     String hostName = null;
     String portNumber = null;
     String homeName = null;
     if (args.length >= 1)
        hostName = args[0];
     if (args.length >=2)
        portNumber = args[1];
     if (args.length >=3)
        homeName = args[2];

     PmiClient pmiClnt = new PmiClient(hostName, portNumber, homeName);

     // Root of PMI data tree
     CpdCollection rootCol = pmiClnt.createRootCollection();
```

Figure 11. Example of PMI client code (Part 1 of 3)

```
     // Set performance descriptor (pd) list
     // pdList will include all PerfDescriptors for data retrieval
     PerfDescriptorList pdList = new PerfDescriptorList();
     try {
          // If you want to query PmiClient to find the PerfDescriptor
          // you need, you can go through listNodes, listServers, and
          // listMembers to list all the PerfDescriptors and extract
          // the one you want.
          PerfDescriptor[] nodePds = pmiClnt.listNodes();
          String nodeName = nodePds[0].getName();
          System.out.println("after listNodes:" + nodeName);
          PerfDescriptor[] serverPds = pmiClnt.listServers(
            nodePds[0].getName());
          System.out.println("after listServers");

          if (serverPds == null || serverPds.length == 0) {
               System.out.println("NO app server in node");
               return;
          }

         // For a simple test, get from the first server
          PerfDescriptor[] myPds = pmiClnt.listMembers(serverPds[0]);
          // You can add all pds to PerfDescriptorList
          for (int i = 0; i < myPds.length; i++) {
            if (myPds[i].getModuleName().equals(
                "com.ibm.websphere.pmi.beanModule")
            || myPds[i].getModuleName().equals(
                "com.ibm.websphere.pmi.connectionPoolModule")
            || myPds[i].getModuleName.equals(
                "com.ibm.websphere.pmi.webAppModule"))
               pdList.addDescriptor(myPds[i]);
          }

          // Or, if you know the data path you want, you can create your own
          String[] thisPath = new String[]{"thisNode", "thisServer",
               "com.ibm.websphere.pmi.transactionModule"};
          // Suppose you are interested only in dataIds 1, 2, and 3
          PerfDescriptor thisPd = pmiClnt.createPerfDescriptor(thisPath,
               new int[]{1, 2, 3});
          pdList.addDescriptor(thisPd);

} catch (Exception ex) {
               System.out.println("Exception calling CollectorAE");
               ex.printStackTrack();
     }
```

Figure 11. Example of PMI client code (Part 2 of 3)

```
     // Retrieve the data in pdList
     CpdCollection[] cpdCols = null;
     try {
          for (int i = 0; i < 10; i++) {
               java.lang.Thread.sleep(1000);
               cpdCols = pmiClnt.gets(pdList, true);
               if (cpdCols == null || cpdCols.length == 0) {
                    System.out.println(
                      "PMI data return null--possible wrong pds");
               }
               for (int j = 0; j < cpdCols.length; j=++) {
                    rootCol.update(cpdCols[j]);
                    report(cpdCols[j]);
               }
          }
     } catch (Exception ex {
               System.out.println("Exception to call thread sleep");
     }
   }

   // Simple method to make sure we are getting the correct CpdCollection
   private static void report(CpdCollection col) {
        System.out.println("\n\n");
        if (col == null) {
             System.out.println("report: null CpdCollection");
             return;
        }
        System.out.println("report--CpdCollection ");
        printPD(col.getDescriptor());
        CpdData[] dataMembers = col.dataMembers();
        if (dataMembers != null) {
             System.out.println("report CpdCollection: dataMembers is " +
                  dataMembers.length);
             for (int i = 0; i < dataMembers.length; i++) {
                  CpdData data = dataMembers[i];
                  printPD(data.getDescriptor());
             }

        }
        CpdCollection[] subCollections = col.subcollections();
        if (subCollections != null) {
             for (int i = 0; i < subCollections.length; i++) {
                  report(subCollections[i]);
             }
        }
   }

   // Simple method to write the full name of a pd
   private static void printPD(PerfDescriptor pd) {
        System.out.println(pd.getFullName());
   }
}
```

Figure 11. Example of PMI client code (Part 3 of 3)

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS DOCUMENT "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the document. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

For Component Broker:
IBM Corporation
Department LZKS
11400 Burnet Road
Austin, TX 78758
U.S.A.

For TXSeries:
IBM Corporation
ATTN: Software Licensing
11 Stanwix Street
Pittsburgh, PA 15222
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks and service marks

The following terms are trademarks or registered trademarks of the IBM Corporation in the United States, other countries, or both:

Advanced Peer-to-Peer Networking
AFS
AIX
APPN
AS/400
CICS
CICS OS/2
CICS/400
CICS/6000
CICS/ESA
CICS/MVS
CICS/VSE
CICSPlex
DB2
DCE Encina Lightweight Client
DFS
Encina
IBM
IBM System Application Architecture
IMS
IMS/ESA
Language Environment
MQSeries
MVS/ESA
NetView
Open Class
OS/2
OS/390
OS/400
Parallel Sysplex
PowerPC
RACF
RAMAO
RMF
RISC System/6000
RS/6000
S/390
SAA
SecureWay
TeamConnection
Transarc
TXSeries
VSE/ESA
VTAM
VisualAge
WebSphere

Domino, Lotus, and LotusScript are trademarks or registered trademarks of Lotus Development Corporation in the United States, other countries, or both.

Tivoli is a registered trademark of Tivoli Systems, Inc. in the United States, other countries, or both.

ActiveX, Microsoft, Visual Basic, Visual C++, Visual J++, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Some of this documentation is based on material from Object Management Group bearing the following copyright notices:

Copyright 1995, 1996 AT&T/NCR
Copyright 1995, 1996 BNR Europe Ltd.
Copyright 1991, 1992, 1995, 1996 by Digital Equipment Corporation
Copyright 1996 Gradient Technologies, Inc.
Copyright 1995, 1996 Groupe Bull
Copyright 1995, 1996 Expersoft Corporation
Copyright 1996 FUJITSU LIMITED
Copyright 1996 Genesis Development Corporation
Copyright 1989, 1990, 1991, 1992, 1995, 1996 by Hewlett-Packard Company
Copyright 1991, 1992, 1995, 1996 by HyperDesk Corporation
Copyright 1995, 1996 IBM Corporation
Copyright 1995, 1996 ICL, plc
Copyright 1995, 1996 Ing. C. Olivetti &C.Sp
Copyright 1997 International Computers Limited
Copyright 1995, 1996 IONA Technologies, Ltd.
Copyright 1995, 1996 Itasca Systems, Inc.
Copyright 1991, 1992, 1995, 1996 by NCR Corporation
Copyright 1997 Netscape Communications Corporation
Copyright 1997 Northern Telecom Limited
Copyright 1995, 1996 Novell USG
Copyright 1995, 1996 02 Technolgies
Copyright 1991, 1992, 1995, 1996 by Object Design, Inc.
Copyright 1991, 1992, 1995, 1996 Object Management Group, Inc.
Copyright 1995, 1996 Objectivity, Inc.
Copyright 1995, 1996 Oracle Corporation
Copyright 1995, 1996 Persistence Software

Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

This software contains RSA encryption code.

Other company, product, and service names may be trademarks or service marks of others.