****
1779 Fri May 3 09:27:51 2013
new/make/java/management/Exportedfiles.gmk
****

```
1 #
2 # Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
2 # Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
3 # DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
4 #
5 # This code is free software; you can redistribute it and/or modify it

7 # under the terms of the GNU General Public License version 2 only, as
8 # published by the Free Software Foundation.  Oracle designates this
9 # particular file as subject to the "Classpath" exception as provided
10 # by Oracle in the LICENSE file that accompanied this code.
11 #
12 # This code is distributed in the hope that it will be useful, but WITHOUT
13 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
14 # FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
15 # version 2 for more details (a copy is included in the LICENSE file that
16 # accompanied this code).
17 #
18 # You should have received a copy of the GNU General Public License version
19 # 2 along with this work; if not, write to the Free Software Foundation,
20 # Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
21 #
22 # Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
23 # or visit www.oracle.com if you need additional information or have any
24 # questions.
25 #

27 #
28 # These are the names of Java classes for which we will make .h files.
29 #

31 FILES_export = \
32         sun/management/ClassLoadingImpl.java \
33         sun/management/DiagnosticCommandImpl.java \
34         sun/management/FileSystemImpl.java \
35         sun/management/Flag.java \
36         sun/management/GarbageCollectorImpl.java \
37         sun/management/GcInfoBuilder.java \
38         sun/management/HotSpotDiagnostic.java \
39         sun/management/HotspotThread.java \
40         sun/management/MemoryImpl.java \
41         sun/management/MemoryManagerImpl.java \
42         sun/management/MemoryPoolImpl.java \
43         sun/management/ThreadImpl.java \
44         sun/management/VMManagementImpl.java
```

****
1479 Fri May 3 09:27:52 2013
new/make/java/management/FILES_c.gmk
****

```
  1 #
  2 # Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
  2 # Copyright (c) 2003, 2005, Oracle and/or its affiliates. All rights reserved.
  3 # DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
  4 #
  5 # This code is free software; you can redistribute it and/or modify it
  6 # under the terms of the GNU General Public License version 2 only, as
  7 # published by the Free Software Foundation.  Oracle designates this
  8 # particular file as subject to the "Classpath" exception as provided
  9 # by Oracle in the LICENSE file that accompanied this code.
 10 #
 11 # This code is distributed in the hope that it will be useful, but WITHOUT
 12 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 13 # FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 14 # version 2 for more details (a copy is included in the LICENSE file that
 15 # accompanied this code).
 16 #
 17 # You should have received a copy of the GNU General Public License version
 18 # 2 along with this work; if not, write to the Free Software Foundation,
 19 # Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 20 #
 21 # Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 22 # or visit www.oracle.com if you need additional information or have any
 23 # questions.
 24 #

 26 FILES_c = \
 27         ClassLoadingImpl.c \
 28         DiagnosticCommandImpl.c \
 29         FileSystemImpl.c \
 30         Flag.c \
 31         GarbageCollectorImpl.c \
 32         GcInfoBuilder.c \
 33         HotSpotDiagnostic.c \
 34         HotspotThread.c \
 35         MemoryImpl.c \
 36         MemoryManagerImpl.c \
 37         MemoryPoolImpl.c \
 38         ThreadImpl.c \
 39         VMManagementImpl.c \
 40         management.c
```

****
6954 Fri May 3 09:27:55 2013
new/make/java/management/mapfile-vers
****

```
  1 #
  2 # Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
  2 # Copyright (c) 2005, 2012, Oracle and/or its affiliates. All rights reserved.
  3 # DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
  4 #
  5 # This code is free software; you can redistribute it and/or modify it
  6 # under the terms of the GNU General Public License version 2 only, as
  7 # published by the Free Software Foundation.  Oracle designates this
  8 # particular file as subject to the "Classpath" exception as provided
  9 # by Oracle in the LICENSE file that accompanied this code.
 10 #
 11 # This code is distributed in the hope that it will be useful, but WITHOUT
 12 # ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 13 # FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 14 # version 2 for more details (a copy is included in the LICENSE file that
 15 # accompanied this code).
 16 #
 17 # You should have received a copy of the GNU General Public License version
 18 # 2 along with this work; if not, write to the Free Software Foundation,
 19 # Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 20 #
 21 # Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 22 # or visit www.oracle.com if you need additional information or have any
 23 # questions.
 24 #

 26 # Define library interface.

 28 SUNWprivate_1.1 {
 29         global:
 30             Java_com_sun_management_UnixOperatingSystem_getCommittedVirtualMemor
 31             Java_com_sun_management_UnixOperatingSystem_getFreePhysicalMemorySiz
 32             Java_com_sun_management_UnixOperatingSystem_getFreeSwapSpaceSize;
 33             Java_com_sun_management_UnixOperatingSystem_getMaxFileDescriptorCoun
 34             Java_com_sun_management_UnixOperatingSystem_getOpenFileDescriptorCou
 35             Java_com_sun_management_UnixOperatingSystem_getProcessCpuLoad;
 36             Java_com_sun_management_UnixOperatingSystem_getProcessCpuTime;
 37             Java_com_sun_management_UnixOperatingSystem_getSystemCpuLoad;
 38             Java_com_sun_management_UnixOperatingSystem_getTotalPhysicalMemorySi
 39             Java_com_sun_management_UnixOperatingSystem_getTotalSwapSpaceSize;
 40             Java_com_sun_management_UnixOperatingSystem_initialize;
 41             Java_sun_management_ClassLoadingImpl_setVerboseClass;
 42             Java_sun_management_DiagnosticCommandImpl_executeDiagnosticCommand;
 43             Java_sun_management_DiagnosticCommandImpl_getDiagnosticCommands;
 44             Java_sun_management_DiagnosticCommandImpl_getDiagnosticCommandInfo;
 45             Java_sun_management_DiagnosticCommandImpl_setNotificationEnabled;
 46             Java_sun_management_FileSystemImpl_isAccessUserOnly0;
 47             Java_sun_management_Flag_getAllFlagNames;
 48             Java_sun_management_Flag_getFlags;
 49             Java_sun_management_Flag_getInternalFlagCount;
 50             Java_sun_management_Flag_initialize;
 51             Java_sun_management_Flag_setLongValue;
 52             Java_sun_management_Flag_setBooleanValue;
 53             Java_sun_management_Flag_setStringValue;
 54             Java_sun_management_GarbageCollectorImpl_getCollectionCount;
 55             Java_sun_management_GarbageCollectorImpl_getCollectionTime;
 56             Java_sun_management_GarbageCollectorImpl_setNotificationEnabled;
 57             Java_sun_management_GcInfoBuilder_fillGcAttributeInfo;
 58             Java_sun_management_GcInfoBuilder_getLastGcInfo0;
 59             Java_sun_management_GcInfoBuilder_getNumGcExtAttributes;
 60             Java_sun_management_HotSpotDiagnostic_dumpHeap;
 61             Java_sun_management_HotspotThread_getInternalThreadCount;
```



```
 62             Java_sun_management_HotspotThread_getInternalThreadTimes0;
 63             Java_sun_management_MemoryImpl_getMemoryManagers0;
 64             Java_sun_management_MemoryImpl_getMemoryPools0;
 65             Java_sun_management_MemoryImpl_getMemoryUsage0;
 66             Java_sun_management_MemoryImpl_setVerboseGC;
 67             Java_sun_management_MemoryManagerImpl_getMemoryPools0;
 68             Java_sun_management_MemoryPoolImpl_getCollectionUsage0;
 69             Java_sun_management_MemoryPoolImpl_getMemoryManagers0;
 70             Java_sun_management_MemoryPoolImpl_getPeakUsage0;
 71             Java_sun_management_MemoryPoolImpl_getUsage0;
 72             Java_sun_management_MemoryPoolImpl_resetPeakUsage0;
 73             Java_sun_management_MemoryPoolImpl_setCollectionThreshold0;
 74             Java_sun_management_MemoryPoolImpl_setPoolCollectionSensor;
 75             Java_sun_management_MemoryPoolImpl_setPoolUsageSensor;
 76             Java_sun_management_MemoryPoolImpl_setUsageThreshold0;
 77             Java_sun_management_ThreadImpl_dumpThreads0;
 78             Java_sun_management_ThreadImpl_findDeadlockedThreads0;
 79             Java_sun_management_ThreadImpl_findMonitorDeadlockedThreads0;
 80             Java_sun_management_ThreadImpl_getThreadInfo1;
 81             Java_sun_management_ThreadImpl_getThreads;
 82             Java_sun_management_ThreadImpl_getThreadTotalCpuTime0;
 83             Java_sun_management_ThreadImpl_getThreadTotalCpuTime1;
 84             Java_sun_management_ThreadImpl_getThreadUserCpuTime0;
 85             Java_sun_management_ThreadImpl_getThreadUserCpuTime1;
 86             Java_sun_management_ThreadImpl_getThreadAllocatedMemory1;
 87             Java_sun_management_ThreadImpl_resetContentionTimes0;
 88             Java_sun_management_ThreadImpl_resetPeakThreadCount0;
 89             Java_sun_management_ThreadImpl_setThreadContentionMonitoringEnabled0
 90             Java_sun_management_ThreadImpl_setThreadCpuTimeEnabled0;
 91             Java_sun_management_ThreadImpl_setThreadAllocatedMemoryEnabled0;;
 87             Java_sun_management_ThreadImpl_setThreadAllocatedMemoryEnabled0;
 92             Java_sun_management_VMManagementImpl_getAvailableProcessors;
 93             Java_sun_management_VMManagementImpl_getClassInitializationTime;
 94             Java_sun_management_VMManagementImpl_getClassLoadingTime;
 95             Java_sun_management_VMManagementImpl_getClassVerificationTime;
 96             Java_sun_management_VMManagementImpl_getDaemonThreadCount;
 97             Java_sun_management_VMManagementImpl_getInitializedClassCount;
 98             Java_sun_management_VMManagementImpl_getLiveThreadCount;
 99             Java_sun_management_VMManagementImpl_getLoadedClassSize;
100             Java_sun_management_VMManagementImpl_getMethodDataSize;
101             Java_sun_management_VMManagementImpl_getPeakThreadCount;
102             Java_sun_management_VMManagementImpl_getProcessId;
103             Java_sun_management_VMManagementImpl_getSafepointCount;
104             Java_sun_management_VMManagementImpl_getSafepointSyncTime;
105             Java_sun_management_VMManagementImpl_getStartupTime;
106             Java_sun_management_VMManagementImpl_getTotalApplicationNonStoppedTi
107             Java_sun_management_VMManagementImpl_getTotalClassCount;
108             Java_sun_management_VMManagementImpl_getTotalCompileTime;
109             Java_sun_management_VMManagementImpl_getTotalSafepointTime;
110             Java_sun_management_VMManagementImpl_getTotalThreadCount;
111             Java_sun_management_VMManagementImpl_getUnloadedClassCount;
112             Java_sun_management_VMManagementImpl_getUnloadedClassSize;
113             Java_sun_management_VMManagementImpl_getVerboseClass;
114             Java_sun_management_VMManagementImpl_getVerboseGC;
115             Java_sun_management_VMManagementImpl_getVersion0;
116             Java_sun_management_VMManagementImpl_getVmArguments0;
117             Java_sun_management_VMManagementImpl_initOptionalSupportFields;
118             Java_sun_management_VMManagementImpl_isThreadContentionMonitoringEna
119             Java_sun_management_VMManagementImpl_isThreadCpuTimeEnabled;
120             Java_sun_management_VMManagementImpl_isThreadAllocatedMemoryEnabled;
121             JNI_OnLoad;
122         local:
123             *;
124 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    6960 Fri May  3 09:27:56 2013
new/makefiles/mapfiles/libmanagement/mapfile-vers
**********************************************************
```

```
#
# Copyright (c) 2005, 2013, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2005, 2012, Oracle and/or its affiliates. All rights reserved.
# DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
#
# This code is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License version 2 only, as
# published by the Free Software Foundation.  Oracle designates this
# particular file as subject to the "Classpath" exception as provided
# by Oracle in the LICENSE file that accompanied this code.
#
# This code is distributed in the hope that it will be useful, but WITHOUT
# ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
# version 2 for more details (a copy is included in the LICENSE file that
# accompanied this code).
#
# You should have received a copy of the GNU General Public License version
# 2 along with this work; if not, write to the Free Software Foundation,
# Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
#
# Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
# or visit www.oracle.com if you need additional information or have any
# questions.
#

# Define library interface.

SUNWprivate_1.1 {
        global:
            Java_com_sun_management_UnixOperatingSystem_getCommittedVirtualMemor
            Java_com_sun_management_UnixOperatingSystem_getFreePhysicalMemorySiz
            Java_com_sun_management_UnixOperatingSystem_getFreeSwapSpaceSize;
            Java_com_sun_management_UnixOperatingSystem_getMaxFileDescriptorCoun
            Java_com_sun_management_UnixOperatingSystem_getOpenFileDescriptorCou
            Java_com_sun_management_UnixOperatingSystem_getProcessCpuLoad;
            Java_com_sun_management_UnixOperatingSystem_getProcessCpuTime;
            Java_com_sun_management_UnixOperatingSystem_getSystemCpuLoad;
            Java_com_sun_management_UnixOperatingSystem_getTotalPhysicalMemorySi
            Java_com_sun_management_UnixOperatingSystem_getTotalSwapSpaceSize;
            Java_com_sun_management_UnixOperatingSystem_initialize;
            Java_sun_management_ClassLoadingImpl_setVerboseClass;
            Java_sun_management_DiagnosticCommandImpl_executeDiagnosticCommand;
            Java_sun_management_DiagnosticCommandImpl_getDiagnosticCommands;
            Java_sun_management_DiagnosticCommandImpl_getDiagnosticCommandInfo;
            Java_sun_management_DiagnosticCommandImpl_setNotificationEnabled;
            Java_sun_management_FileSystemImpl_isAccessUserOnly0;
            Java_sun_management_Flag_getAllFlagNames;
            Java_sun_management_Flag_getFlags;
            Java_sun_management_Flag_getInternalFlagCount;
            Java_sun_management_Flag_initialize;
            Java_sun_management_Flag_setLongValue;
            Java_sun_management_Flag_setBooleanValue;
            Java_sun_management_Flag_setStringValue;
            Java_sun_management_GarbageCollectorImpl_getCollectionCount;
            Java_sun_management_GarbageCollectorImpl_getCollectionTime;
            Java_sun_management_GarbageCollectorImpl_setNotificationEnabled;
            Java_sun_management_GcInfoBuilder_fillGcAttributeInfo;
            Java_sun_management_GcInfoBuilder_getLastGcInfo0;
            Java_sun_management_GcInfoBuilder_getNumGcExtAttributes;
            Java_sun_management_HotSpotDiagnostic_dumpHeap;
            Java_sun_management_HotspotThread_getInternalThreadCount;
```



```
            Java_sun_management_HotspotThread_getInternalThreadTimes0;
            Java_sun_management_MemoryImpl_getMemoryManagers0;
            Java_sun_management_MemoryImpl_getMemoryPools0;
            Java_sun_management_MemoryImpl_getMemoryUsage0;
            Java_sun_management_MemoryImpl_setVerboseGC;
            Java_sun_management_MemoryManagerImpl_getMemoryPools0;
            Java_sun_management_MemoryPoolImpl_getCollectionUsage0;
            Java_sun_management_MemoryPoolImpl_getMemoryManagers0;
            Java_sun_management_MemoryPoolImpl_getPeakUsage0;
            Java_sun_management_MemoryPoolImpl_getUsage0;
            Java_sun_management_MemoryPoolImpl_resetPeakUsage0;
            Java_sun_management_MemoryPoolImpl_setCollectionThreshold0;
            Java_sun_management_MemoryPoolImpl_setPoolCollectionSensor;
            Java_sun_management_MemoryPoolImpl_setPoolUsageSensor;
            Java_sun_management_MemoryPoolImpl_setUsageThreshold0;
            Java_sun_management_ThreadImpl_dumpThreads0;
            Java_sun_management_ThreadImpl_findDeadlockedThreads0;
            Java_sun_management_ThreadImpl_findMonitorDeadlockedThreads0;
            Java_sun_management_ThreadImpl_getThreadInfo1;
            Java_sun_management_ThreadImpl_getThreads;
            Java_sun_management_ThreadImpl_getThreadTotalCpuTime0;
            Java_sun_management_ThreadImpl_getThreadTotalCpuTime1;
            Java_sun_management_ThreadImpl_getThreadUserCpuTime0;
            Java_sun_management_ThreadImpl_getThreadUserCpuTime1;
            Java_sun_management_ThreadImpl_getThreadAllocatedMemory1;
            Java_sun_management_ThreadImpl_resetContentionTimes0;
            Java_sun_management_ThreadImpl_resetPeakThreadCount0;
            Java_sun_management_ThreadImpl_setThreadContentionMonitoringEnabled0
            Java_sun_management_ThreadImpl_setThreadCpuTimeEnabled0;
            Java_sun_management_ThreadImpl_setThreadAllocatedMemoryEnabled0;
            Java_sun_management_VMManagementImpl_getAvailableProcessors;
            Java_sun_management_VMManagementImpl_getClassInitializationTime;
            Java_sun_management_VMManagementImpl_getClassLoadingTime;
            Java_sun_management_VMManagementImpl_getClassVerificationTime;
            Java_sun_management_VMManagementImpl_getDaemonThreadCount;
            Java_sun_management_VMManagementImpl_getInitializedClassCount;
            Java_sun_management_VMManagementImpl_getLiveThreadCount;
            Java_sun_management_VMManagementImpl_getLoadedClassSize;
            Java_sun_management_VMManagementImpl_getMethodDataSize;
            Java_sun_management_VMManagementImpl_getPeakThreadCount;
            Java_sun_management_VMManagementImpl_getProcessId;
            Java_sun_management_VMManagementImpl_getSafepointCount;
            Java_sun_management_VMManagementImpl_getSafepointSyncTime;
            Java_sun_management_VMManagementImpl_getStartupTime;
            Java_sun_management_VMManagementImpl_getTotalApplicationNonStoppedTi
            Java_sun_management_VMManagementImpl_getTotalClassCount;
            Java_sun_management_VMManagementImpl_getTotalCompileTime;
            Java_sun_management_VMManagementImpl_getTotalSafepointTime;
            Java_sun_management_VMManagementImpl_getTotalThreadCount;
            Java_sun_management_VMManagementImpl_getUnloadedClassCount;
            Java_sun_management_VMManagementImpl_getUnloadedClassSize;
            Java_sun_management_VMManagementImpl_getVerboseClass;
            Java_sun_management_VMManagementImpl_getVerboseGC;
            Java_sun_management_VMManagementImpl_getVersion0;
            Java_sun_management_VMManagementImpl_getVmArguments0;
            Java_sun_management_VMManagementImpl_initOptionalSupportFields;
            Java_sun_management_VMManagementImpl_isThreadContentionMonitoringEna
            Java_sun_management_VMManagementImpl_isThreadCpuTimeEnabled;
            Java_sun_management_VMManagementImpl_isThreadAllocatedMemoryEnabled;
            JNI_OnLoad;
        local:
            *;
};
______unchanged_portion_omitted_
```

```
**********************************************************
   34435 Fri May  3 09:27:58 2013
new/src/share/classes/java/lang/management/ManagementFactory.java
**********************************************************
```

```
   1 /*
   2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
   2  * Copyright (c) 2003, 2012, Oracle and/or its affiliates. All rights reserved.
   3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
   4  *
   5  * This code is free software; you can redistribute it and/or modify it
   6  * under the terms of the GNU General Public License version 2 only, as
   7  * published by the Free Software Foundation.  Oracle designates this
   8  * particular file as subject to the "Classpath" exception as provided
   9  * by Oracle in the LICENSE file that accompanied this code.
  10  *
  11  * This code is distributed in the hope that it will be useful, but WITHOUT
  12  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
  13  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
  14  * version 2 for more details (a copy is included in the LICENSE file that
  15  * accompanied this code).
  16  *
  17  * You should have received a copy of the GNU General Public License version
  18  * 2 along with this work; if not, write to the Free Software Foundation,
  19  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
  20  *
  21  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
  22  * or visit www.oracle.com if you need additional information or have any
  23  * questions.
  24  */

  26 package java.lang.management;
  27 import javax.management.DynamicMBean;
  28 import javax.management.MBeanServer;
  29 import javax.management.MBeanServerConnection;
  30 import javax.management.MBeanServerFactory;
  31 import javax.management.MBeanServerPermission;
  32 import javax.management.MXBean;
  33 import javax.management.NotificationEmitter;
  34 import javax.management.ObjectInstance;
  35 import javax.management.ObjectName;
  36 import javax.management.InstanceAlreadyExistsException;
  37 import javax.management.InstanceNotFoundException;
  38 import javax.management.MalformedObjectNameException;
  39 import javax.management.MBeanRegistrationException;
  40 import javax.management.NotCompliantMBeanException;
  41 import javax.management.StandardEmitterMBean;
  42 import javax.management.StandardMBean;
  43 import java.util.Collections;
  44 import java.util.List;
  45 import java.util.Set;
  46 import java.util.HashMap;
  47 import java.util.HashSet;
  48 import java.security.AccessController;
  49 import java.security.Permission;
  50 import java.security.PrivilegedAction;
  51 import java.security.PrivilegedActionException;
  52 import java.security.PrivilegedExceptionAction;
  53 import javax.management.JMX;
  54 import sun.management.ManagementFactoryHelper;

  56 /**
  57  * The {@code ManagementFactory} class is a factory class for getting
  58  * managed beans for the Java platform.
  59  * This class consists of static methods each of which returns
  60  * one or more <i>platform MXBeans</i> representing
  61  * the management interface of a component of the Java virtual
```



```
  62  * machine.
  63  * <p>
  64  * <h4><a name="MXBean">Platform MXBeans</a></h4>
  65  * <p>
  66  * A platform MXBean is a <i>managed bean</i> that
  67  * conforms to the <a href="../../../javax/management/package-summary.html">JMX<
  68  * Instrumentation Specification and only uses a set of basic data types.
  69  * A JMX management application and the {@linkplain
  70  * #getPlatformMBeanServer platform MBeanServer}
  71  * can interoperate without requiring classes for MXBean specific
  72  * data types.
  73  * The data types being transmitted between the JMX connector
  74  * server and the connector client are
  75  * {@linkplain javax.management.openmbean.OpenType open types}
  76  * and this allows interoperation across versions.
  77  * See <a href="../../../javax/management/MXBean.html#MXBean-spec">
  78  * the specification of MXBeans</a> for details.
  79  *
  80  * <a name="MXBeanNames"></a>
  81  * <p>Each platform MXBean is a {@link PlatformManagedObject}
  82  * and it has a unique
  83  * {@link javax.management.ObjectName ObjectName} for
  84  * registration in the platform {@code MBeanServer} as returned by
  85  * by the {@link PlatformManagedObject#getObjectName getObjectName}
  86  * method.
  87  *
  88  * <p>
  89  * An application can access a platform MXBean in the following ways:
  90  * <h5>1. Direct access to an MXBean interface</h5>
  91  * <blockquote>
  92  * <ul>
  93  *     <li>Get an MXBean instance by calling the
  94  *         {@link #getPlatformMXBean(Class) getPlatformMXBean} or
  95  *         {@link #getPlatformMXBeans(Class) getPlatformMXBeans} method
  96  *         and access the MXBean locally in the running
  97  *         virtual machine.
  98  *         </li>
  99  *     <li>Construct an MXBean proxy instance that forwards the
 100  *         method calls to a given {@link MBeanServer MBeanServer} by calling
 101  *         the {@link #getPlatformMXBean(MBeanServerConnection, Class)} or
 102  *         {@link #getPlatformMXBeans(MBeanServerConnection, Class)} method.
 103  *         The {@link #newPlatformMXBeanProxy newPlatformMXBeanProxy} method
 104  *         can also be used to construct an MXBean proxy instance of
 105  *         a given {@code ObjectName}.
 106  *         A proxy is typically constructed to remotely access
 107  *         an MXBean of another running virtual machine.
 108  *         </li>
 109  * </ul>
 110  * <h5>2. Indirect access to an MXBean interface via MBeanServer</h5>
 111  * <ul>
 112  *     <li>Go through the platform {@code MBeanServer} to access MXBeans
 113  *         locally or a specific <tt>MBeanServerConnection</tt> to access
 114  *         MXBeans remotely.
 115  *         The attributes and operations of an MXBean use only
 116  *         <em>JMX open types</em> which include basic data types,
 117  *         {@link javax.management.openmbean.CompositeData CompositeData},
 118  *         and {@link javax.management.openmbean.TabularData TabularData}
 119  *         defined in
 120  *         {@link javax.management.openmbean.OpenType OpenType}.
 121  *         The mapping is specified in
 122  *         the {@linkplain javax.management.MXBean MXBean} specification
 123  *         for details.
 124  *        </li>
 125  * </ul>
 126  * </blockquote>
 127  *
```

```
 128  * <p>
 129  * The {@link #getPlatformManagementInterfaces getPlatformManagementInterfaces}
 130  * method returns all management interfaces supported in the Java virtual machin
 131  * including the standard management interfaces listed in the tables
 132  * below as well as the management interfaces extended by the JDK implementation
 133  * <p>
 134  * A Java virtual machine has a single instance of the following management
 135  * interfaces:
 136  *
 137  * <blockquote>
 138  * <table border>
 139  * <tr>
 140  * <th>Management Interface</th>
 141  * <th>ObjectName</th>
 142  * </tr>
 143  * <tr>
 144  * <td> {@link ClassLoadingMXBean} </td>
 145  * <td> {@link #CLASS_LOADING_MXBEAN_NAME
 146  *             java.lang:type=ClassLoading}</td>
 147  * </tr>
 148  * <tr>
 149  * <td> {@link MemoryMXBean} </td>
 150  * <td> {@link #MEMORY_MXBEAN_NAME
 151  *             java.lang:type=Memory}</td>
 152  * </tr>
 153  * <tr>
 154  * <td> {@link ThreadMXBean} </td>
 155  * <td> {@link #THREAD_MXBEAN_NAME
 156  *             java.lang:type=Threading}</td>
 157  * </tr>
 158  * <tr>
 159  * <td> {@link RuntimeMXBean} </td>
 160  * <td> {@link #RUNTIME_MXBEAN_NAME
 161  *             java.lang:type=Runtime}</td>
 162  * </tr>
 163  * <tr>
 164  * <td> {@link OperatingSystemMXBean} </td>
 165  * <td> {@link #OPERATING_SYSTEM_MXBEAN_NAME
 166  *             java.lang:type=OperatingSystem}</td>
 167  * </tr>
 168  * <tr>
 169  * <td> {@link PlatformLoggingMXBean} </td>
 170  * <td> {@link java.util.logging.LogManager#LOGGING_MXBEAN_NAME
 171  *             java.util.logging:type=Logging}</td>
 172  * </tr>
 173  * </table>
 174  * </blockquote>
 175  *
 176  * <p>
 177  * A Java virtual machine has zero or a single instance of
 178  * the following management interfaces.
 179  *
 180  * <blockquote>
 181  * <table border>
 182  * <tr>
 183  * <th>Management Interface</th>
 184  * <th>ObjectName</th>
 185  * </tr>
 186  * <tr>
 187  * <td> {@link CompilationMXBean} </td>
 188  * <td> {@link #COMPILATION_MXBEAN_NAME
 189  *             java.lang:type=Compilation}</td>
 190  * </tr>
 191  * </table>
 192  * </blockquote>
 193  *
```



```
 194  * <p>
 195  * A Java virtual machine may have one or more instances of the following
 196  * management interfaces.
 197  * <blockquote>
 198  * <table border>
 199  * <tr>
 200  * <th>Management Interface</th>
 201  * <th>ObjectName</th>
 202  * </tr>
 203  * <tr>
 204  * <td> {@link GarbageCollectorMXBean} </td>
 205  * <td> {@link #GARBAGE_COLLECTOR_MXBEAN_DOMAIN_TYPE
 206  *             java.lang:type=GarbageCollector}<tt>,name=</tt><i>collector's nam
 207  * </tr>
 208  * <tr>
 209  * <td> {@link MemoryManagerMXBean} </td>
 210  * <td> {@link #MEMORY_MANAGER_MXBEAN_DOMAIN_TYPE
 211  *             java.lang:type=MemoryManager}<tt>,name=</tt><i>manager's name</i>
 212  * </tr>
 213  * <tr>
 214  * <td> {@link MemoryPoolMXBean} </td>
 215  * <td> {@link #MEMORY_POOL_MXBEAN_DOMAIN_TYPE
 216  *             java.lang:type=MemoryPool}<tt>,name=</tt><i>pool's name</i></td>
 217  * </tr>
 218  * <tr>
 219  * <td> {@link BufferPoolMXBean} </td>
 220  * <td> {@code java.nio:type=BufferPool,name=}<i>pool name</i></td>
 221  * </tr>
 222  * </table>
 223  * </blockquote>
 224  *
 225  * @see <a href="../../../javax/management/package-summary.html">
 226  *      JMX Specification</a>
 227  * @see <a href="package-summary.html#examples">
 228  *      Ways to Access Management Metrics</a>
 229  * @see javax.management.MXBean
 230  *
 231  * @author  Mandy Chung
 232  * @since   1.5
 233  */
 234 public class ManagementFactory {
 235     // A class with only static fields and methods.
 236     private ManagementFactory() {};

 238     /**
 239      * String representation of the
 240      * <tt>ObjectName</tt> for the {@link ClassLoadingMXBean}.
 241      */
 242     public final static String CLASS_LOADING_MXBEAN_NAME =
 243         "java.lang:type=ClassLoading";

 245     /**
 246      * String representation of the
 247      * <tt>ObjectName</tt> for the {@link CompilationMXBean}.
 248      */
 249     public final static String COMPILATION_MXBEAN_NAME =
 250         "java.lang:type=Compilation";

 252     /**
 253      * String representation of the
 254      * <tt>ObjectName</tt> for the {@link MemoryMXBean}.
 255      */
 256     public final static String MEMORY_MXBEAN_NAME =
 257         "java.lang:type=Memory";

 259     /**
```

```
260      * String representation of the
261      * <tt>ObjectName</tt> for the {@link OperatingSystemMXBean}.
262      */
263     public final static String OPERATING_SYSTEM_MXBEAN_NAME =
264         "java.lang:type=OperatingSystem";

266     /**
267      * String representation of the
268      * <tt>ObjectName</tt> for the {@link RuntimeMXBean}.
269      */
270     public final static String RUNTIME_MXBEAN_NAME =
271         "java.lang:type=Runtime";

273     /**
274      * String representation of the
275      * <tt>ObjectName</tt> for the {@link ThreadMXBean}.
276      */
277     public final static String THREAD_MXBEAN_NAME =
278         "java.lang:type=Threading";

280     /**
281      * The domain name and the type key property in
282      * the <tt>ObjectName</tt> for a {@link GarbageCollectorMXBean}.
283      * The unique <tt>ObjectName</tt> for a <tt>GarbageCollectorMXBean</tt>
284      * can be formed by appending this string with
285      * "<tt>,name=</tt><i>collector's name</i>".
286      */
287     public final static String GARBAGE_COLLECTOR_MXBEAN_DOMAIN_TYPE =
288         "java.lang:type=GarbageCollector";

290     /**
291      * The domain name and the type key property in
292      * the <tt>ObjectName</tt> for a {@link MemoryManagerMXBean}.
293      * The unique <tt>ObjectName</tt> for a <tt>MemoryManagerMXBean</tt>
294      * can be formed by appending this string with
295      * "<tt>,name=</tt><i>manager's name</i>".
296      */
297     public final static String MEMORY_MANAGER_MXBEAN_DOMAIN_TYPE=
298         "java.lang:type=MemoryManager";

300     /**
301      * The domain name and the type key property in
302      * the <tt>ObjectName</tt> for a {@link MemoryPoolMXBean}.
303      * The unique <tt>ObjectName</tt> for a <tt>MemoryPoolMXBean</tt>
304      * can be formed by appending this string with
305      * <tt>,name=</tt><i>pool's name</i>.
306      */
307     public final static String MEMORY_POOL_MXBEAN_DOMAIN_TYPE=
308         "java.lang:type=MemoryPool";

310     /**
311      * Returns the managed bean for the class loading system of
312      * the Java virtual machine.
313      *
314      * @return a {@link ClassLoadingMXBean} object for
315      * the Java virtual machine.
316      */
317     public static ClassLoadingMXBean getClassLoadingMXBean() {
318         return ManagementFactoryHelper.getClassLoadingMXBean();
319     }

321     /**
322      * Returns the managed bean for the memory system of
323      * the Java virtual machine.
324      *
325      * @return a {@link MemoryMXBean} object for the Java virtual machine.
```



```
326      */
327     public static MemoryMXBean getMemoryMXBean() {
328         return ManagementFactoryHelper.getMemoryMXBean();
329     }

331     /**
332      * Returns the managed bean for the thread system of
333      * the Java virtual machine.
334      *
335      * @return a {@link ThreadMXBean} object for the Java virtual machine.
336      */
337     public static ThreadMXBean getThreadMXBean() {
338         return ManagementFactoryHelper.getThreadMXBean();
339     }

341     /**
342      * Returns the managed bean for the runtime system of
343      * the Java virtual machine.
344      *
345      * @return a {@link RuntimeMXBean} object for the Java virtual machine.

347      */
348     public static RuntimeMXBean getRuntimeMXBean() {
349         return ManagementFactoryHelper.getRuntimeMXBean();
350     }

352     /**
353      * Returns the managed bean for the compilation system of
354      * the Java virtual machine.  This method returns <tt>null</tt>
355      * if the Java virtual machine has no compilation system.
356      *
357      * @return a {@link CompilationMXBean} object for the Java virtual
358      *   machine or <tt>null</tt> if the Java virtual machine has
359      *   no compilation system.
360      */
361     public static CompilationMXBean getCompilationMXBean() {
362         return ManagementFactoryHelper.getCompilationMXBean();
363     }

365     /**
366      * Returns the managed bean for the operating system on which
367      * the Java virtual machine is running.
368      *
369      * @return an {@link OperatingSystemMXBean} object for
370      * the Java virtual machine.
371      */
372     public static OperatingSystemMXBean getOperatingSystemMXBean() {
373         return ManagementFactoryHelper.getOperatingSystemMXBean();
374     }

376     /**
377      * Returns a list of {@link MemoryPoolMXBean} objects in the
378      * Java virtual machine.
379      * The Java virtual machine can have one or more memory pools.
380      * It may add or remove memory pools during execution.
381      *
382      * @return a list of <tt>MemoryPoolMXBean</tt> objects.
383      *
384      */
385     public static List<MemoryPoolMXBean> getMemoryPoolMXBeans() {
386         return ManagementFactoryHelper.getMemoryPoolMXBeans();
387     }

389     /**
390      * Returns a list of {@link MemoryManagerMXBean} objects
391      * in the Java virtual machine.
```

```
392      * The Java virtual machine can have one or more memory managers.
393      * It may add or remove memory managers during execution.
394      *
395      * @return a list of <tt>MemoryManagerMXBean</tt> objects.
396      *
397      */
398     public static List<MemoryManagerMXBean> getMemoryManagerMXBeans() {
399         return ManagementFactoryHelper.getMemoryManagerMXBeans();
400     }


403     /**
404      * Returns a list of {@link GarbageCollectorMXBean} objects
405      * in the Java virtual machine.
406      * The Java virtual machine may have one or more
407      * <tt>GarbageCollectorMXBean</tt> objects.
408      * It may add or remove <tt>GarbageCollectorMXBean</tt>
409      * during execution.
410      *
411      * @return a list of <tt>GarbageCollectorMXBean</tt> objects.
412      *
413      */
414     public static List<GarbageCollectorMXBean> getGarbageCollectorMXBeans() {
415         return ManagementFactoryHelper.getGarbageCollectorMXBeans();
416     }

418     private static MBeanServer platformMBeanServer;
419     /**
420      * Returns the platform {@link javax.management.MBeanServer MBeanServer}.
421      * On the first call to this method, it first creates the platform
422      * {@code MBeanServer} by calling the
423      * {@link javax.management.MBeanServerFactory#createMBeanServer
424      * MBeanServerFactory.createMBeanServer}
425      * method and registers each platform MXBean in this platform
426      * {@code MBeanServer} with its
427      * {@link PlatformManagedObject#getObjectName ObjectName}.
428      * This method, in subsequent calls, will simply return the
429      * initially created platform {@code MBeanServer}.
430      * <p>
431      * MXBeans that get created and destroyed dynamically, for example,
432      * memory {@link MemoryPoolMXBean pools} and
433      * {@link MemoryManagerMXBean managers},
434      * will automatically be registered and deregistered into the platform
435      * {@code MBeanServer}.
436      * <p>
437      * If the system property {@code javax.management.builder.initial}
438      * is set, the platform {@code MBeanServer} creation will be done
439      * by the specified {@link javax.management.MBeanServerBuilder}.
440      * <p>
441      * It is recommended that this platform MBeanServer also be used
442      * to register other application managed beans
443      * besides the platform MXBeans.
444      * This will allow all MBeans to be published through the same
445      * {@code MBeanServer} and hence allow for easier network publishing
446      * and discovery.
447      * Name conflicts with the platform MXBeans should be avoided.
448      *
449      * @return the platform {@code MBeanServer}; the platform
450      *         MXBeans are registered into the platform {@code MBeanServer}
451      *         at the first time this method is called.
452      *
453      * @exception SecurityException if there is a security manager
454      * and the caller does not have the permission required by
455      * {@link javax.management.MBeanServerFactory#createMBeanServer}.
456      *
457      * @see javax.management.MBeanServerFactory
```



```
458      * @see javax.management.MBeanServerFactory#createMBeanServer
459      */
460     public static synchronized MBeanServer getPlatformMBeanServer() {
461         SecurityManager sm = System.getSecurityManager();
462         if (sm != null) {
463             Permission perm = new MBeanServerPermission("createMBeanServer");
464             sm.checkPermission(perm);
465         }

467         if (platformMBeanServer == null) {
468             platformMBeanServer = MBeanServerFactory.createMBeanServer();
469             for (PlatformComponent pc : PlatformComponent.values()) {
470                 List<? extends PlatformManagedObject> list =
471                     pc.getMXBeans(pc.getMXBeanInterface());
472                 for (PlatformManagedObject o : list) {
473                     // Each PlatformComponent represents one management
474                     // interface. Some MXBean may extend another one.
475                     // The MXBean instances for one platform component
476                     // (returned by pc.getMXBeans()) might be also
477                     // the MXBean instances for another platform component.
478                     // e.g. com.sun.management.GarbageCollectorMXBean
479                     //
480                     // So need to check if an MXBean instance is registered
481                     // before registering into the platform MBeanServer
482                     if (!platformMBeanServer.isRegistered(o.getObjectName())) {
483                         addMXBean(platformMBeanServer, o);
484                     }
485                 }
486             }
487             HashMap<ObjectName, DynamicMBean> dynmbeans =
488                     ManagementFactoryHelper.getPlatformDynamicMBeans();
489             for (ObjectName on : dynmbeans.keySet()) {
490                 addDynamicMBean(platformMBeanServer, dynmbeans.get(on), on);
491             }
492         }
493         return platformMBeanServer;
494     }

496     /**
497      * Returns a proxy for a platform MXBean interface of a
498      * given <a href="#MXBeanNames">MXBean name</a>
499      * that forwards its method calls through the given
500      * <tt>MBeanServerConnection</tt>.
501      *
502      * <p>This method is equivalent to:
503      * <blockquote>
504      * {@link java.lang.reflect.Proxy#newProxyInstance
505      *        Proxy.newProxyInstance}<tt>(mxbeanInterface.getClassLoader(),
506      *        new Class[] { mxbeanInterface }, handler)</tt>
507      * </blockquote>
508      *
509      * where <tt>handler</tt> is an {@link java.lang.reflect.InvocationHandler
510      * InvocationHandler} to which method invocations to the MXBean interface
511      * are dispatched. This <tt>handler</tt> converts an input parameter
512      * from an MXBean data type to its mapped open type before forwarding
513      * to the <tt>MBeanServer</tt> and converts a return value from
514      * an MXBean method call through the <tt>MBeanServer</tt>
515      * from an open type to the corresponding return type declared in
516      * the MXBean interface.
517      *
518      * <p>
519      * If the MXBean is a notification emitter (i.e.,
520      * it implements
521      * {@link javax.management.NotificationEmitter NotificationEmitter}),
522      * both the <tt>mxbeanInterface</tt> and <tt>NotificationEmitter</tt>
523      * will be implemented by this proxy.
```

```
524      *
525      * <p>
526      * <b>Notes:</b>
527      * <ol>
528      * <li>Using an MXBean proxy is a convenience remote access to
529      * a platform MXBean of a running virtual machine.  All method
530      * calls to the MXBean proxy are forwarded to an
531      * <tt>MBeanServerConnection</tt> where
532      * {@link java.io.IOException IOException} may be thrown
533      * when the communication problem occurs with the connector server.
534      * An application remotely accesses the platform MXBeans using
535      * proxy should prepare to catch <tt>IOException</tt> as if
536      * accessing with the <tt>MBeanServerConnector</tt> interface.</li>
537      *
538      * <li>When a client application is designed to remotely access MXBeans
539      * for a running virtual machine whose version is different than
540      * the version on which the application is running,
541      * it should prepare to catch
542      * {@link java.io.InvalidObjectException InvalidObjectException}
543      * which is thrown when an MXBean proxy receives a name of an
544      * enum constant which is missing in the enum class loaded in
545      * the client application. </li>
546      *
547      * <li>{@link javax.management.MBeanServerInvocationHandler
548      * MBeanServerInvocationHandler} or its
549      * {@link javax.management.MBeanServerInvocationHandler#newProxyInstance
550      * newProxyInstance} method cannot be used to create
551      * a proxy for a platform MXBean. The proxy object created
552      * by <tt>MBeanServerInvocationHandler</tt> does not handle
553      * the properties of the platform MXBeans described in
554      * the <a href="#MXBean">class specification</a>.
555      *</li>
556      * </ol>
557      *
558      * @param connection the <tt>MBeanServerConnection</tt> to forward to.
559      * @param mxbeanName the name of a platform MXBean within
560      * <tt>connection</tt> to forward to. <tt>mxbeanName</tt> must be
561      * in the format of {@link ObjectName ObjectName}.
562      * @param mxbeanInterface the MXBean interface to be implemented
563      * by the proxy.
564      *
565      * @throws IllegalArgumentException if
566      * <ul>
567      * <li><tt>mxbeanName</tt> is not with a valid
568      *     {@link ObjectName ObjectName} format, or</li>
569      * <li>the named MXBean in the <tt>connection</tt> is
570      *     not a MXBean provided by the platform, or</li>
571      * <li>the named MXBean is not registered in the
572      *     <tt>MBeanServerConnection</tt>, or</li>
573      * <li>the named MXBean is not an instance of the given
574      *     <tt>mxbeanInterface</tt></li>
575      * </ul>
576      *
577      * @throws java.io.IOException if a communication problem
578      * occurred when accessing the <tt>MBeanServerConnection</tt>.
579      */
580     public static <T> T
581         newPlatformMXBeanProxy(MBeanServerConnection connection,
582                                String mxbeanName,
583                                Class<T> mxbeanInterface)
584             throws java.io.IOException {

586         // Only allow MXBean interfaces from rt.jar loaded by the
587         // bootstrap class loader
588         final Class<?> cls = mxbeanInterface;
589         ClassLoader loader =
```



```
590             AccessController.doPrivileged(new PrivilegedAction<ClassLoader>() {
591                 public ClassLoader run() {
592                     return cls.getClassLoader();
593                 }
594             });
595         if (!sun.misc.VM.isSystemDomainLoader(loader)) {
596             throw new IllegalArgumentException(mxbeanName +
597                 " is not a platform MXBean");
598         }

600         try {
601             final ObjectName objName = new ObjectName(mxbeanName);
602             // skip the isInstanceOf check for LoggingMXBean
603             String intfName = mxbeanInterface.getName();
604             if (!connection.isInstanceOf(objName, intfName)) {
605                 throw new IllegalArgumentException(mxbeanName +
606                     " is not an instance of " + mxbeanInterface);
607             }

609             final Class[] interfaces;
610             // check if the registered MBean is a notification emitter
611             boolean emitter = connection.isInstanceOf(objName, NOTIF_EMITTER);

613             // create an MXBean proxy
614             return JMX.newMXBeanProxy(connection, objName, mxbeanInterface,
615                                       emitter);
616         } catch (InstanceNotFoundException|MalformedObjectNameException e) {
617             throw new IllegalArgumentException(e);
618         }
619     }

621     /**
622      * Returns the platform MXBean implementing
623      * the given {@code mxbeanInterface} which is specified
624      * to have one single instance in the Java virtual machine.
625      * This method may return {@code null} if the management interface
626      * is not implemented in the Java virtual machine (for example,
627      * a Java virtual machine with no compilation system does not
628      * implement {@link CompilationMXBean});
629      * otherwise, this method is equivalent to calling:
630      * <pre>
631      *    {@link #getPlatformMXBeans(Class)
632      *      getPlatformMXBeans(mxbeanInterface)}.get(0);
633      * </pre>
634      *
635      * @param mxbeanInterface a management interface for a platform
636      *     MXBean with one single instance in the Java virtual machine
637      *     if implemented.
638      *
639      * @return the platform MXBean that implements
640      * {@code mxbeanInterface}, or {@code null} if not exist.
641      *
642      * @throws IllegalArgumentException if {@code mxbeanInterface}
643      * is not a platform management interface or
644      * not a singleton platform MXBean.
645      *
646      * @since 1.7
647      */
648     public static <T extends PlatformManagedObject>
649             T getPlatformMXBean(Class<T> mxbeanInterface) {
650         PlatformComponent pc = PlatformComponent.getPlatformComponent(mxbeanInte
651         if (pc == null)
652             throw new IllegalArgumentException(mxbeanInterface.getName() +
653                 " is not a platform management interface");
654         if (!pc.isSingleton())
655             throw new IllegalArgumentException(mxbeanInterface.getName() +
```

```
656                 " can have zero or more than one instances");
657
658         return pc.getSingletonMXBean(mxbeanInterface);
659     }
660
661     /**
662      * Returns the list of platform MXBeans implementing
663      * the given {@code mxbeanInterface} in the Java
664      * virtual machine.
665      * The returned list may contain zero, one, or more instances.
666      * The number of instances in the returned list is defined
667      * in the specification of the given management interface.
668      * The order is undefined and there is no guarantee that
669      * the list returned is in the same order as previous invocations.
670      *
671      * @param mxbeanInterface a management interface for a platform
672      *                        MXBean
673      *
674      * @return the list of platform MXBeans that implement
675      * {@code mxbeanInterface}.
676      *
677      * @throws IllegalArgumentException if {@code mxbeanInterface}
678      * is not a platform management interface.
679      *
680      * @since 1.7
681      */
682     public static <T extends PlatformManagedObject> List<T>
683             getPlatformMXBeans(Class<T> mxbeanInterface) {
684         PlatformComponent pc = PlatformComponent.getPlatformComponent(mxbeanInte
685         if (pc == null)
686             throw new IllegalArgumentException(mxbeanInterface.getName() +
687                 " is not a platform management interface");
688         return Collections.unmodifiableList(pc.getMXBeans(mxbeanInterface));
689     }
690
691     /**
692      * Returns the platform MXBean proxy for
693      * {@code mxbeanInterface} which is specified to have one single
694      * instance in a Java virtual machine and the proxy will
695      * forward the method calls through the given {@code MBeanServerConnection}.
696      * This method may return {@code null} if the management interface
697      * is not implemented in the Java virtual machine being monitored
698      * (for example, a Java virtual machine with no compilation system
699      * does not implement {@link CompilationMXBean});
700      * otherwise, this method is equivalent to calling:
701      * <pre>
702      *     {@link #getPlatformMXBeans(MBeanServerConnection, Class)
703      *         getPlatformMXBeans(connection, mxbeanInterface)}.get(0);
704      * </pre>
705      *
706      * @param connection the {@code MBeanServerConnection} to forward to.
707      * @param mxbeanInterface a management interface for a platform
708      *     MXBean with one single instance in the Java virtual machine
709      *     being monitored, if implemented.
710      *
711      * @return the platform MXBean proxy for
712      * forwarding the method calls of the {@code mxbeanInterface}
713      * through the given {@code MBeanServerConnection},
714      * or {@code null} if not exist.
715      *
716      * @throws IllegalArgumentException if {@code mxbeanInterface}
717      * is not a platform management interface or
718      * not a singleton platform MXBean.
719      * @throws java.io.IOException if a communication problem
720      * occurred when accessing the {@code MBeanServerConnection}.
721      *
```



```
722      * @see #newPlatformMXBeanProxy
723      * @since 1.7
724      */
725     public static <T extends PlatformManagedObject>
726             T getPlatformMXBean(MBeanServerConnection connection,
727                                 Class<T> mxbeanInterface)
728         throws java.io.IOException
729     {
730         PlatformComponent pc = PlatformComponent.getPlatformComponent(mxbeanInte
731         if (pc == null)
732             throw new IllegalArgumentException(mxbeanInterface.getName() +
733                 " is not a platform management interface");
734         if (!pc.isSingleton())
735             throw new IllegalArgumentException(mxbeanInterface.getName() +
736                 " can have zero or more than one instances");
737         return pc.getSingletonMXBean(connection, mxbeanInterface);
738     }
739
740     /**
741      * Returns the list of the platform MXBean proxies for
742      * forwarding the method calls of the {@code mxbeanInterface}
743      * through the given {@code MBeanServerConnection}.
744      * The returned list may contain zero, one, or more instances.
745      * The number of instances in the returned list is defined
746      * in the specification of the given management interface.
747      * The order is undefined and there is no guarantee that
748      * the list returned is in the same order as previous invocations.
749      *
750      * @param connection the {@code MBeanServerConnection} to forward to.
751      * @param mxbeanInterface a management interface for a platform
752      *                        MXBean
753      *
754      * @return the list of platform MXBean proxies for
755      * forwarding the method calls of the {@code mxbeanInterface}
756      * through the given {@code MBeanServerConnection}.
757      *
758      * @throws IllegalArgumentException if {@code mxbeanInterface}
759      * is not a platform management interface.
760      *
761      * @throws java.io.IOException if a communication problem
762      * occurred when accessing the {@code MBeanServerConnection}.
763      *
764      * @see #newPlatformMXBeanProxy
765      * @since 1.7
766      */
767     public static <T extends PlatformManagedObject>
768             List<T> getPlatformMXBeans(MBeanServerConnection connection,
769                                        Class<T> mxbeanInterface)
770         throws java.io.IOException
771     {
772         PlatformComponent pc = PlatformComponent.getPlatformComponent(mxbeanInte
773         if (pc == null) {
774             throw new IllegalArgumentException(mxbeanInterface.getName() +
775                 " is not a platform management interface");
776         }
777         return Collections.unmodifiableList(pc.getMXBeans(connection, mxbeanInte
778     }
779
780     /**
781      * Returns the set of {@code Class} objects, subinterface of
782      * {@link PlatformManagedObject}, representing
783      * all management interfaces for
784      * monitoring and managing the Java platform.
785      *
786      * @return the set of {@code Class} objects, subinterface of
787      * {@link PlatformManagedObject} representing
```

```
788      * the management interfaces for
789      * monitoring and managing the Java platform.
790      *
791      * @since 1.7
792      */
793     public static Set<Class<? extends PlatformManagedObject>>
794             getPlatformManagementInterfaces()
795     {
796         Set<Class<? extends PlatformManagedObject>> result =
797             new HashSet<>();
798         for (PlatformComponent component: PlatformComponent.values()) {
799             result.add(component.getMXBeanInterface());
800         }
801         return Collections.unmodifiableSet(result);
802     }

804     private static final String NOTIF_EMITTER =
805         "javax.management.NotificationEmitter";

807     /**
808      * Registers an MXBean.
809      */
810     private static void addMXBean(final MBeanServer mbs, final PlatformManagedOb
811         // Make DynamicMBean out of MXBean by wrapping it with a StandardMBean
812         try {
813             AccessController.doPrivileged(new PrivilegedExceptionAction<Void>()
814                 public Void run() throws InstanceAlreadyExistsException,
815                                          MBeanRegistrationException,
816                                          NotCompliantMBeanException {
817                     final DynamicMBean dmbean;
818                     if (pmo instanceof DynamicMBean) {
819                         dmbean = DynamicMBean.class.cast(pmo);
820                     } else if (pmo instanceof NotificationEmitter) {
821                         dmbean = new StandardEmitterMBean(pmo, null, true, (Noti
822                     } else {
823                         dmbean = new StandardMBean(pmo, null, true);
824                     }

826                     mbs.registerMBean(dmbean, pmo.getObjectName());
827                     return null;
828                 }
829             });
830         } catch (PrivilegedActionException e) {
831             throw new RuntimeException(e.getException());
832         }
833     }

835     /**
836      * Registers a DynamicMBean.
837      */
838     private static void addDynamicMBean(final MBeanServer mbs,
839                                         final DynamicMBean dmbean,
840                                         final ObjectName on) {
841         try {
842             AccessController.doPrivileged(new PrivilegedExceptionAction<Void>()
843                 @Override
844                 public Void run() throws InstanceAlreadyExistsException,
845                                          MBeanRegistrationException,
846                                          NotCompliantMBeanException {
847                     mbs.registerMBean(dmbean, on);
848                     return null;
849                 }
850             });
851         } catch (PrivilegedActionException e) {
852             throw new RuntimeException(e.getException());
853         }
```



```
854     }
855 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   17288 Fri May  3 09:27:59 2013
new/src/share/classes/sun/management/ManagementFactoryHelper.java
**********************************************************
```

```
   1 /*
   2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
   2  * Copyright (c) 2003, 2012, Oracle and/or its affiliates. All rights reserved.
   3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
   4  *
   5  * This code is free software; you can redistribute it and/or modify it
   6  * under the terms of the GNU General Public License version 2 only, as
   7  * published by the Free Software Foundation.  Oracle designates this
   8  * particular file as subject to the "Classpath" exception as provided
   9  * by Oracle in the LICENSE file that accompanied this code.
  10  *
  11  * This code is distributed in the hope that it will be useful, but WITHOUT
  12  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
  13  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
  14  * version 2 for more details (a copy is included in the LICENSE file that
  15  * accompanied this code).
  16  *
  17  * You should have received a copy of the GNU General Public License version
  18  * 2 along with this work; if not, write to the Free Software Foundation,
  19  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
  20  *
  21  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
  22  * or visit www.oracle.com if you need additional information or have any
  23  * questions.
  24  */

  26 package sun.management;

  28 import java.lang.management.*;

  30 import javax.management.DynamicMBean;
  31 import javax.management.InstanceAlreadyExistsException;
  32 import javax.management.InstanceNotFoundException;
  33 import javax.management.MBeanServer;
  34 import javax.management.MBeanRegistrationException;
  35 import javax.management.NotCompliantMBeanException;
  36 import javax.management.ObjectName;
  37 import javax.management.RuntimeOperationsException;
  38 import java.security.AccessController;
  39 import java.security.PrivilegedActionException;
  40 import java.security.PrivilegedExceptionAction;

  42 import sun.util.logging.LoggingSupport;

  44 import java.util.ArrayList;
  45 import java.util.Collections;
  46 import java.util.HashMap;
  47 import java.util.List;
  48 import com.sun.management.DiagnosticCommandMBean;
  49 import com.sun.management.OSMBeanFactory;
  50 import com.sun.management.HotSpotDiagnosticMXBean;

  52 import static java.lang.management.ManagementFactory.*;

  54 /**
  55  * ManagementFactoryHelper provides static factory methods to create
  56  * instances of the management interface.
  57  */
  58 public class ManagementFactoryHelper {
  59     private ManagementFactoryHelper() {};

  61     private static VMManagement jvm;
```



```
  63     private static ClassLoadingImpl    classMBean = null;
  64     private static MemoryImpl          memoryMBean = null;
  65     private static ThreadImpl          threadMBean = null;
  66     private static RuntimeImpl         runtimeMBean = null;
  67     private static CompilationImpl     compileMBean = null;
  68     private static OperatingSystemImpl osMBean = null;

  70     public static synchronized ClassLoadingMXBean getClassLoadingMXBean() {
  71         if (classMBean == null) {
  72             classMBean = new ClassLoadingImpl(jvm);
  73         }
  74         return classMBean;
  75     }

  77     public static synchronized MemoryMXBean getMemoryMXBean() {
  78         if (memoryMBean == null) {
  79             memoryMBean = new MemoryImpl(jvm);
  80         }
  81         return memoryMBean;
  82     }

  84     public static synchronized ThreadMXBean getThreadMXBean() {
  85         if (threadMBean == null) {
  86             threadMBean = new ThreadImpl(jvm);
  87         }
  88         return threadMBean;
  89     }

  91     public static synchronized RuntimeMXBean getRuntimeMXBean() {
  92         if (runtimeMBean == null) {
  93             runtimeMBean = new RuntimeImpl(jvm);
  94         }
  95         return runtimeMBean;
  96     }

  98     public static synchronized CompilationMXBean getCompilationMXBean() {
  99         if (compileMBean == null && jvm.getCompilerName() != null) {
 100             compileMBean = new CompilationImpl(jvm);
 101         }
 102         return compileMBean;
 103     }

 105     public static synchronized OperatingSystemMXBean getOperatingSystemMXBean()
 106         if (osMBean == null) {
 107             osMBean = (OperatingSystemImpl)
 108                       OSMBeanFactory.getOperatingSystemMXBean(jvm);
 109         }
 110         return osMBean;
 111     }

 113     public static List<MemoryPoolMXBean> getMemoryPoolMXBeans() {
 114         MemoryPoolMXBean[] pools = MemoryImpl.getMemoryPools();
 115         List<MemoryPoolMXBean> list = new ArrayList<>(pools.length);
 116         for (MemoryPoolMXBean p : pools) {
 117             list.add(p);
 118         }
 119         return list;
 120     }

 122     public static List<MemoryManagerMXBean> getMemoryManagerMXBeans() {
 123         MemoryManagerMXBean[]  mgrs = MemoryImpl.getMemoryManagers();
 124         List<MemoryManagerMXBean> result = new ArrayList<>(mgrs.length);
 125         for (MemoryManagerMXBean m : mgrs) {
 126             result.add(m);
 127         }
```

```
128         return result;
129     }

131     public static List<GarbageCollectorMXBean> getGarbageCollectorMXBeans() {
132         MemoryManagerMXBean[]  mgrs = MemoryImpl.getMemoryManagers();
133         List<GarbageCollectorMXBean> result = new ArrayList<>(mgrs.length);
134         for (MemoryManagerMXBean m : mgrs) {
135             if (GarbageCollectorMXBean.class.isInstance(m)) {
136                 result.add(GarbageCollectorMXBean.class.cast(m));
137             }
138         }
139         return result;
140     }

142     public static PlatformLoggingMXBean getPlatformLoggingMXBean() {
143         if (LoggingSupport.isAvailable()) {
144             return PlatformLoggingImpl.instance;
145         } else {
146             return null;
147         }
148     }

150     // The logging MXBean object is an instance of
151     // PlatformLoggingMXBean and java.util.logging.LoggingMXBean
152     // but it can't directly implement two MXBean interfaces
153     // as a compliant MXBean implements exactly one MXBean interface,
154     // or if it implements one interface that is a subinterface of
155     // all the others; otherwise, it is a non-compliant MXBean
156     // and MBeanServer will throw NotCompliantMBeanException.
157     // See the Definition of an MXBean section in javax.management.MXBean spec.
158     //
159     // To create a compliant logging MXBean, define a LoggingMXBean interface
160     // that extend PlatformLoggingMXBean and j.u.l.LoggingMXBean
161     interface LoggingMXBean
162         extends PlatformLoggingMXBean, java.util.logging.LoggingMXBean {
163     }

165     static class PlatformLoggingImpl implements LoggingMXBean
166     {
167         final static PlatformLoggingMXBean instance = new PlatformLoggingImpl();
168         final static String LOGGING_MXBEAN_NAME = "java.util.logging:type=Loggin

170         private volatile ObjectName objname;  // created lazily
171         @Override
172         public ObjectName getObjectName() {
173             ObjectName result = objname;
174             if (result == null) {
175                 synchronized (this) {
176                     result = objname;
177                     if (result == null) {
178                         result = Util.newObjectName(LOGGING_MXBEAN_NAME);
179                         objname = result;
180                     }
181                 }
182             }
183             return result;
184         }

186         @Override
187         public java.util.List<String> getLoggerNames() {
188             return LoggingSupport.getLoggerNames();
189         }

191         @Override
192         public String getLoggerLevel(String loggerName) {
193             return LoggingSupport.getLoggerLevel(loggerName);
```



```
194         }

196         @Override
197         public void setLoggerLevel(String loggerName, String levelName) {
198             LoggingSupport.setLoggerLevel(loggerName, levelName);
199         }

201         @Override
202         public String getParentLoggerName(String loggerName) {
203             return LoggingSupport.getParentLoggerName(loggerName);
204         }
205     }

207     private static List<BufferPoolMXBean> bufferPools = null;
208     public static synchronized List<BufferPoolMXBean> getBufferPoolMXBeans() {
209         if (bufferPools == null) {
210             bufferPools = new ArrayList<>(2);
211             bufferPools.add(createBufferPoolMXBean(sun.misc.SharedSecrets.getJav
212                 .getDirectBufferPool()));
213             bufferPools.add(createBufferPoolMXBean(sun.nio.ch.FileChannelImpl
214                 .getMappedBufferPool()));
215         }
216         return bufferPools;
217     }

219     private final static String BUFFER_POOL_MXBEAN_NAME = "java.nio:type=BufferP

221     /**
222      * Creates management interface for the given buffer pool.
223      */
224     private static BufferPoolMXBean
225         createBufferPoolMXBean(final sun.misc.JavaNioAccess.BufferPool pool)
226     {
227         return new BufferPoolMXBean() {
228             private volatile ObjectName objname;  // created lazily
229             @Override
230             public ObjectName getObjectName() {
231                 ObjectName result = objname;
232                 if (result == null) {
233                     synchronized (this) {
234                         result = objname;
235                         if (result == null) {
236                             result = Util.newObjectName(BUFFER_POOL_MXBEAN_NAME
237                                 ",name=" + pool.getName());
238                             objname = result;
239                         }
240                     }
241                 }
242                 return result;
243             }
244             @Override
245             public String getName() {
246                 return pool.getName();
247             }
248             @Override
249             public long getCount() {
250                 return pool.getCount();
251             }
252             @Override
253             public long getTotalCapacity() {
254                 return pool.getTotalCapacity();
255             }
256             @Override
257             public long getMemoryUsed() {
258                 return pool.getMemoryUsed();
259             }
```

```
260         };
261     }

263     private static HotSpotDiagnostic hsDiagMBean = null;
264     private static HotspotRuntime hsRuntimeMBean = null;
265     private static HotspotClassLoading hsClassMBean = null;
266     private static HotspotThread hsThreadMBean = null;
267     private static HotspotCompilation hsCompileMBean = null;
268     private static HotspotMemory hsMemoryMBean = null;
269     private static DiagnosticCommandImpl hsDiagCommandMBean = null;

271     public static synchronized HotSpotDiagnosticMXBean getDiagnosticMXBean() {
272         if (hsDiagMBean == null) {
273             hsDiagMBean = new HotSpotDiagnostic();
274         }
275         return hsDiagMBean;
276     }

278     /**
279      * This method is for testing only.
280      */
281     public static synchronized HotspotRuntimeMBean getHotspotRuntimeMBean() {
282         if (hsRuntimeMBean == null) {
283             hsRuntimeMBean = new HotspotRuntime(jvm);
284         }
285         return hsRuntimeMBean;
286     }

288     /**
289      * This method is for testing only.
290      */
291     public static synchronized HotspotClassLoadingMBean getHotspotClassLoadingMB
292         if (hsClassMBean == null) {
293             hsClassMBean = new HotspotClassLoading(jvm);
294         }
295         return hsClassMBean;
296     }

298     /**
299      * This method is for testing only.
300      */
301     public static synchronized HotspotThreadMBean getHotspotThreadMBean() {
302         if (hsThreadMBean == null) {
303             hsThreadMBean = new HotspotThread(jvm);
304         }
305         return hsThreadMBean;
306     }

308     /**
309      * This method is for testing only.
310      */
311     public static synchronized HotspotMemoryMBean getHotspotMemoryMBean() {
312         if (hsMemoryMBean == null) {
313             hsMemoryMBean = new HotspotMemory(jvm);
314         }
315         return hsMemoryMBean;
316     }

318     public static synchronized DiagnosticCommandMBean getDiagnosticCommandMBean(
319         // Remote Diagnostic Commands may not be supported
320         if (hsDiagCommandMBean == null && jvm.isRemoteDiagnosticCommandsSupporte
321             hsDiagCommandMBean = new DiagnosticCommandImpl(jvm);
322         }
323         return hsDiagCommandMBean;
324     }
```



```
326     /**
327      * This method is for testing only.
328      */
329     public static synchronized HotspotCompilationMBean getHotspotCompilationMBea
330         if (hsCompileMBean == null) {
331             hsCompileMBean = new HotspotCompilation(jvm);
332         }
333         return hsCompileMBean;
334     }

336     /**
337      * Registers a given MBean if not registered in the MBeanServer;
338      * otherwise, just return.
339      */
340     private static void addMBean(MBeanServer mbs, Object mbean, String mbeanName
341         try {
342             final ObjectName objName = Util.newObjectName(mbeanName);

344             // inner class requires these fields to be final
345             final MBeanServer mbs0 = mbs;
346             final Object mbean0 = mbean;
347             AccessController.doPrivileged(new PrivilegedExceptionAction<Void>()
348                 public Void run() throws MBeanRegistrationException,
349                                          NotCompliantMBeanException {
350                     try {
351                         mbs0.registerMBean(mbean0, objName);
352                         return null;
353                     } catch (InstanceAlreadyExistsException e) {
354                         // if an instance with the object name exists in
355                         // the MBeanServer ignore the exception
356                     }
357                     return null;
358                 }
359             });
360         } catch (PrivilegedActionException e) {
361             throw Util.newException(e.getException());
362         }
363     }

365     private final static String HOTSPOT_CLASS_LOADING_MBEAN_NAME =
366         "sun.management:type=HotspotClassLoading";

368     private final static String HOTSPOT_COMPILATION_MBEAN_NAME =
369         "sun.management:type=HotspotCompilation";

371     private final static String HOTSPOT_MEMORY_MBEAN_NAME =
372         "sun.management:type=HotspotMemory";

374     private static final String HOTSPOT_RUNTIME_MBEAN_NAME =
375         "sun.management:type=HotspotRuntime";

377     private final static String HOTSPOT_THREAD_MBEAN_NAME =
378         "sun.management:type=HotspotThreading";

380     final static String HOTSPOT_DIAGNOSTIC_COMMAND_MBEAN_NAME =
381         "com.sun.management:type=DiagnosticCommand";

383     public static HashMap<ObjectName, DynamicMBean> getPlatformDynamicMBeans() {
384         HashMap<ObjectName, DynamicMBean> map = new HashMap<>();
385         if (getDiagnosticCommandMBean() != null) {
386             map.put(Util.newObjectName(HOTSPOT_DIAGNOSTIC_COMMAND_MBEAN_NAME), g
387         }
388         return map;
389     }

391     static void registerInternalMBeans(MBeanServer mbs) {
```

```
392         // register all internal MBeans if not registered
393         // No exception is thrown if a MBean with that object name
394         // already registered
395         addMBean(mbs, getHotspotClassLoadingMBean(),
396             HOTSPOT_CLASS_LOADING_MBEAN_NAME);
397         addMBean(mbs, getHotspotMemoryMBean(),
398             HOTSPOT_MEMORY_MBEAN_NAME);
399         addMBean(mbs, getHotspotRuntimeMBean(),
400             HOTSPOT_RUNTIME_MBEAN_NAME);
401         addMBean(mbs, getHotspotThreadMBean(),
402             HOTSPOT_THREAD_MBEAN_NAME);

404         // CompilationMBean may not exist
405         if (getCompilationMXBean() != null) {
406             addMBean(mbs, getHotspotCompilationMBean(),
407                 HOTSPOT_COMPILATION_MBEAN_NAME);
408         }
409     }

411     private static void unregisterMBean(MBeanServer mbs, String mbeanName) {
412         try {
413             final ObjectName objName = Util.newObjectName(mbeanName);

415             // inner class requires these fields to be final
416             final MBeanServer mbs0 = mbs;
417             AccessController.doPrivileged(new PrivilegedExceptionAction<Void>()
418                 public Void run() throws MBeanRegistrationException,
419                                          RuntimeOperationsException  {
420                     try {
421                         mbs0.unregisterMBean(objName);
422                     } catch (InstanceNotFoundException e) {
423                         // ignore exception if not found
424                     }
425                     return null;
426                 }
427             });
428         } catch (PrivilegedActionException e) {
429             throw Util.newException(e.getException());
430         }
431     }

433     static void unregisterInternalMBeans(MBeanServer mbs) {
434         // unregister all internal MBeans
435         unregisterMBean(mbs, HOTSPOT_CLASS_LOADING_MBEAN_NAME);
436         unregisterMBean(mbs, HOTSPOT_MEMORY_MBEAN_NAME);
437         unregisterMBean(mbs, HOTSPOT_RUNTIME_MBEAN_NAME);
438         unregisterMBean(mbs, HOTSPOT_THREAD_MBEAN_NAME);

440         // CompilationMBean may not exist
441         if (getCompilationMXBean() != null) {
442             unregisterMBean(mbs, HOTSPOT_COMPILATION_MBEAN_NAME);
443         }
444     }

446     static {
447         AccessController.doPrivileged(
448             new java.security.PrivilegedAction<Void>() {
449                 public Void run() {
450                     System.loadLibrary("management");
451                     return null;
452                 }
453             });
454         jvm = new VMManagementImpl();
455     }

457     public static boolean isThreadSuspended(int state) {
```



```
458         return ((state & JMM_THREAD_STATE_FLAG_SUSPENDED) != 0);
459     }

461     public static boolean isThreadRunningNative(int state) {
462         return ((state & JMM_THREAD_STATE_FLAG_NATIVE) != 0);
463     }

465     public static Thread.State toThreadState(int state) {
466         // suspended and native bits may be set in state
467         int threadStatus = state & ~JMM_THREAD_STATE_FLAG_MASK;
468         return sun.misc.VM.toThreadState(threadStatus);
469     }

471     // These values are defined in jmm.h
472     private static final int JMM_THREAD_STATE_FLAG_MASK = 0xFFF00000;
473     private static final int JMM_THREAD_STATE_FLAG_SUSPENDED = 0x00100000;
474     private static final int JMM_THREAD_STATE_FLAG_NATIVE = 0x00400000;

476 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4008 Fri May  3 09:28:01 2013
new/src/share/classes/sun/management/VMManagement.java
**********************************************************
```

```
   1 /*
   2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
   2  * Copyright (c) 2003, 2011, Oracle and/or its affiliates. All rights reserved.
   3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
   4  *
   5  * This code is free software; you can redistribute it and/or modify it
   6  * under the terms of the GNU General Public License version 2 only, as
   7  * published by the Free Software Foundation.  Oracle designates this
   8  * particular file as subject to the "Classpath" exception as provided
   9  * by Oracle in the LICENSE file that accompanied this code.
  10  *
  11  * This code is distributed in the hope that it will be useful, but WITHOUT
  12  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
  13  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
  14  * version 2 for more details (a copy is included in the LICENSE file that
  15  * accompanied this code).
  16  *
  17  * You should have received a copy of the GNU General Public License version
  18  * 2 along with this work; if not, write to the Free Software Foundation,
  19  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
  20  *
  21  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
  22  * or visit www.oracle.com if you need additional information or have any
  23  * questions.
  24  */

  26 package sun.management;

  28 import java.util.List;
  29 import sun.management.counter.Counter;
  30 /**
  31  * An interface for the monitoring and management of the
  32  * Java virtual machine.
  33  */
  34 public interface VMManagement {

  36     // Optional supports
  37     public boolean isCompilationTimeMonitoringSupported();
  38     public boolean isThreadContentionMonitoringSupported();
  39     public boolean isThreadContentionMonitoringEnabled();
  40     public boolean isCurrentThreadCpuTimeSupported();
  41     public boolean isOtherThreadCpuTimeSupported();
  42     public boolean isThreadCpuTimeEnabled();
  43     public boolean isBootClassPathSupported();
  44     public boolean isObjectMonitorUsageSupported();
  45     public boolean isSynchronizerUsageSupported();
  46     public boolean isThreadAllocatedMemorySupported();
  47     public boolean isThreadAllocatedMemoryEnabled();
  48     public boolean isGcNotificationSupported();
  49     public boolean isRemoteDiagnosticCommandsSupported();

  51     // Class Loading Subsystem
  52     public long    getTotalClassCount();
  53     public int     getLoadedClassCount();
  54     public long    getUnloadedClassCount();
  55     public boolean getVerboseClass();

  57     // Memory Subsystem
  58     public boolean getVerboseGC();

  60     // Runtime Subsystem
  61     public String  getManagementVersion();
```



```
  62     public String  getVmId();
  63     public String  getVmName();
  64     public String  getVmVendor();
  65     public String  getVmVersion();
  66     public String  getVmSpecName();
  67     public String  getVmSpecVendor();
  68     public String  getVmSpecVersion();
  69     public String  getClassPath();
  70     public String  getLibraryPath();
  71     public String  getBootClassPath();
  72     public List<String> getVmArguments();
  73     public long    getStartupTime();
  74     public int     getAvailableProcessors();

  76     // Compilation Subsystem
  77     public String  getCompilerName();
  78     public long    getTotalCompileTime();

  80     // Thread Subsystem
  81     public long    getTotalThreadCount();
  82     public int     getLiveThreadCount();
  83     public int     getPeakThreadCount();
  84     public int     getDaemonThreadCount();

  86     // Operating System
  87     public String  getOsName();
  88     public String  getOsArch();
  89     public String  getOsVersion();

  91     // Hotspot-specific Runtime support
  92     public long    getSafepointCount();
  93     public long    getTotalSafepointTime();
  94     public long    getSafepointSyncTime();
  95     public long    getTotalApplicationNonStoppedTime();

  97     public long    getLoadedClassSize();
  98     public long    getUnloadedClassSize();
  99     public long    getClassLoadingTime();
 100     public long    getMethodDataSize();
 101     public long    getInitializedClassCount();
 102     public long    getClassInitializationTime();
 103     public long    getClassVerificationTime();

 105     // Performance counter support
 106     public List<Counter>   getInternalCounters(String pattern);
 107 }
______unchanged_portion_omitted_
```

```
**********************************************************
    9129 Fri May  3 09:28:02 2013
new/src/share/classes/sun/management/VMManagementImpl.java
**********************************************************
```

```
 1 /*
 2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
 2  * Copyright (c) 2003, 2011, Oracle and/or its affiliates. All rights reserved.
 3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 4  *
 5  * This code is free software; you can redistribute it and/or modify it
 6  * under the terms of the GNU General Public License version 2 only, as
 7  * published by the Free Software Foundation.  Oracle designates this
 8  * particular file as subject to the "Classpath" exception as provided
 9  * by Oracle in the LICENSE file that accompanied this code.
10  *
11  * This code is distributed in the hope that it will be useful, but WITHOUT
12  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
13  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
14  * version 2 for more details (a copy is included in the LICENSE file that
15  * accompanied this code).
16  *
17  * You should have received a copy of the GNU General Public License version
18  * 2 along with this work; if not, write to the Free Software Foundation,
19  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
20  *
21  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
22  * or visit www.oracle.com if you need additional information or have any
23  * questions.
24  */

26 package sun.management;

28 import sun.misc.Perf;
29 import sun.management.counter.*;
30 import sun.management.counter.perf.*;
31 import java.nio.ByteBuffer;
32 import java.io.IOException;
33 import java.net.InetAddress;
34 import java.net.UnknownHostException;
35 import java.util.List;
36 import java.util.Arrays;
37 import java.util.Collections;
38 import java.security.AccessController;
39 import java.security.PrivilegedAction;
40 import sun.security.action.GetPropertyAction;

42 /**
43  * Implementation of VMManagement interface that accesses the management
44  * attributes and operations locally within the same Java virtual
45  * machine.
46  */
47 class VMManagementImpl implements VMManagement {

49     private static String version;

51     private static boolean compTimeMonitoringSupport;
52     private static boolean threadContentionMonitoringSupport;
53     private static boolean currentThreadCpuTimeSupport;
54     private static boolean otherThreadCpuTimeSupport;
55     private static boolean bootClassPathSupport;
56     private static boolean objectMonitorUsageSupport;
57     private static boolean synchronizerUsageSupport;
58     private static boolean threadAllocatedMemorySupport;
59     private static boolean gcNotificationSupport;
60     private static boolean remoteDiagnosticCommandsSupport;
```



```
63     static {
64         version = getVersion0();
65         if (version == null) {
66             throw new AssertionError("Invalid Management Version");
67         }
68         initOptionalSupportFields();
69     }
70     private native static String getVersion0();
71     private native static void initOptionalSupportFields();

73     // Optional supports
74     public boolean isCompilationTimeMonitoringSupported() {
75         return compTimeMonitoringSupport;
76     }

78     public boolean isThreadContentionMonitoringSupported() {
79         return threadContentionMonitoringSupport;
80     }

82     public boolean isCurrentThreadCpuTimeSupported() {
83         return currentThreadCpuTimeSupport;
84     }

86     public boolean isOtherThreadCpuTimeSupported() {
87         return otherThreadCpuTimeSupport;
88     }

90     public boolean isBootClassPathSupported() {
91         return bootClassPathSupport;
92     }

94     public boolean isObjectMonitorUsageSupported() {
95         return objectMonitorUsageSupport;
96     }

98     public boolean isSynchronizerUsageSupported() {
99         return synchronizerUsageSupport;
100    }

102    public boolean isThreadAllocatedMemorySupported() {
103        return threadAllocatedMemorySupport;
104    }

106    public boolean isGcNotificationSupported() {
107        return gcNotificationSupport;
108    }
109
110    public boolean isRemoteDiagnosticCommandsSupported() {
111        return remoteDiagnosticCommandsSupport;
112    }

114    public native boolean isThreadContentionMonitoringEnabled();
115    public native boolean isThreadCpuTimeEnabled();
116    public native boolean isThreadAllocatedMemoryEnabled();

118    // Class Loading Subsystem
119    public int    getLoadedClassCount() {
120        long count = getTotalClassCount() - getUnloadedClassCount();
121        return (int) count;
122    }
123    public native long getTotalClassCount();
124    public native long getUnloadedClassCount();

126    public native boolean getVerboseClass();
```

```
128     // Memory Subsystem
129     public native boolean getVerboseGC();
130
131     // Runtime Subsystem
132     public String   getManagementVersion() {
133         return version;
134     }
135
136     public String getVmId() {
137         int pid = getProcessId();
138         String hostname = "localhost";
139         try {
140             hostname = InetAddress.getLocalHost().getHostName();
141         } catch (UnknownHostException e) {
142             // ignore
143         }
144
145         return pid + "@" + hostname;
146     }
147     private native int getProcessId();
148
149     public String   getVmName() {
150         return System.getProperty("java.vm.name");
151     }
152
153     public String   getVmVendor() {
154         return System.getProperty("java.vm.vendor");
155     }
156     public String   getVmVersion() {
157         return System.getProperty("java.vm.version");
158     }
159     public String   getVmSpecName()  {
160         return System.getProperty("java.vm.specification.name");
161     }
162     public String   getVmSpecVendor() {
163         return System.getProperty("java.vm.specification.vendor");
164     }
165     public String   getVmSpecVersion() {
166         return System.getProperty("java.vm.specification.version");
167     }
168     public String   getClassPath() {
169         return System.getProperty("java.class.path");
170     }
171     public String   getLibraryPath()  {
172         return System.getProperty("java.library.path");
173     }
174
175     public String   getBootClassPath( ) {
176         PrivilegedAction<String> pa
177             = new GetPropertyAction("sun.boot.class.path");
178         String result =  AccessController.doPrivileged(pa);
179         return result;
180     }
181
182     private List<String> vmArgs = null;
183     public synchronized List<String> getVmArguments() {
184         if (vmArgs == null) {
185             String[] args = getVmArguments0();
186             List<String> l = ((args != null && args.length != 0) ? Arrays.asList
187                                       Collections.<String>emptyList());
188             vmArgs = Collections.unmodifiableList(l);
189         }
190         return vmArgs;
191     }
192     public native String[] getVmArguments0();
```



```
194     public native long getStartupTime();
195     public native int getAvailableProcessors();
196
197     // Compilation Subsystem
198     public String   getCompilerName() {
199         String name =  AccessController.doPrivileged(
200             new PrivilegedAction<String>() {
201                 public String run() {
202                     return System.getProperty("sun.management.compiler");
203                 }
204             });
205         return name;
206     }
207     public native long getTotalCompileTime();
208
209     // Thread Subsystem
210     public native long getTotalThreadCount();
211     public native int  getLiveThreadCount();
212     public native int  getPeakThreadCount();
213     public native int  getDaemonThreadCount();
214
215     // Operating System
216     public String getOsName() {
217         return System.getProperty("os.name");
218     }
219     public String getOsArch() {
220         return System.getProperty("os.arch");
221     }
222     public String getOsVersion() {
223         return System.getProperty("os.version");
224     }
225
226     // Hotspot-specific runtime support
227     public native long getSafepointCount();
228     public native long getTotalSafepointTime();
229     public native long getSafepointSyncTime();
230     public native long getTotalApplicationNonStoppedTime();
231
232     public native long getLoadedClassSize();
233     public native long getUnloadedClassSize();
234     public native long getClassLoadingTime();
235     public native long getMethodDataSize();
236     public native long getInitializedClassCount();
237     public native long getClassInitializationTime();
238     public native long getClassVerificationTime();
239
240     // Performance Counter Support
241     private PerfInstrumentation perfInstr = null;
242     private boolean noPerfData = false;
243
244     private synchronized PerfInstrumentation getPerfInstrumentation() {
245         if (noPerfData || perfInstr != null) {
246             return perfInstr;
247         }
248
249         // construct PerfInstrumentation object
250         Perf perf =  AccessController.doPrivileged(new Perf.GetPerfAction());
251         try {
252             ByteBuffer bb = perf.attach(0, "r");
253             if (bb.capacity() == 0) {
254                 noPerfData = true;
255                 return null;
256             }
257             perfInstr = new PerfInstrumentation(bb);
258         } catch (IllegalArgumentException e) {
259             // If the shared memory doesn't exist e.g. if -XX:-UsePerfData
```

```
260             // was set
261             noPerfData = true;
262         } catch (IOException e) {
263             throw new AssertionError(e);
264         }
265         return perfInstr;
266     }

268     public List<Counter> getInternalCounters(String pattern) {
269         PerfInstrumentation perf = getPerfInstrumentation();
270         if (perf != null) {
271             return perf.findByPattern(pattern);
272         } else {
273             return Collections.emptyList();
274         }
275     }
276 }
```

unchanged_portion_omitted_

****
16931 Fri May 3 09:28:03 2013
new/src/share/javavm/export/jmm.h
****

```
 1 /*
 2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
 2  * Copyright (c) 2003, 2012, Oracle and/or its affiliates. All rights reserved.
 3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 4  *
 5  * This code is free software; you can redistribute it and/or modify it
 6  * under the terms of the GNU General Public License version 2 only, as
 7  * published by the Free Software Foundation.  Oracle designates this
 8  * particular file as subject to the "Classpath" exception as provided
 9  * by Oracle in the LICENSE file that accompanied this code.
10  *
11  * This code is distributed in the hope that it will be useful, but WITHOUT
12  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
13  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
14  * version 2 for more details (a copy is included in the LICENSE file that
15  * accompanied this code).
16  *
17  * You should have received a copy of the GNU General Public License version
18  * 2 along with this work; if not, write to the Free Software Foundation,
19  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
20  *
21  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
22  * or visit www.oracle.com if you need additional information or have any
23  * questions.
24  */

26 #ifndef _JAVA_JMM_H_
27 #define _JAVA_JMM_H_

29 /*
30  * This is a private interface used by JDK for JVM monitoring
31  * and management.
32  *
33  * Bump the version number when either of the following happens:
34  *
35  * 1. There is a change in functions in JmmInterface.
36  *
37  * 2. There is a change in the contract between VM and Java classes.
38  */

40 #include "jni.h"

42 #ifdef __cplusplus
43 extern "C" {
44 #endif

46 enum {
47   JMM_VERSION_1   = 0x20010000,
48   JMM_VERSION_1_0 = 0x20010000,
49   JMM_VERSION_1_1 = 0x20010100, // JDK 6
50   JMM_VERSION_1_2 = 0x20010200, // JDK 7
51   JMM_VERSION_1_2_1 = 0x20010201, // JDK 7 GA
52   JMM_VERSION_1_2_2 = 0x20010202,
53   JMM_VERSION     = 0x20010203
52   JMM_VERSION     = 0x20010202
54 };

56 typedef struct {
57   unsigned int isLowMemoryDetectionSupported : 1;
58   unsigned int isCompilationTimeMonitoringSupported : 1;
59   unsigned int isThreadContentionMonitoringSupported : 1;
60   unsigned int isCurrentThreadCpuTimeSupported : 1;
```



```
61   unsigned int isOtherThreadCpuTimeSupported : 1;
62   unsigned int isBootClassPathSupported : 1;
63   unsigned int isObjectMonitorUsageSupported : 1;
64   unsigned int isSynchronizerUsageSupported : 1;
65   unsigned int isThreadAllocatedMemorySupported : 1;
66   unsigned int isRemoteDiagnosticCommandsSupported : 1;
67   unsigned int : 22;
65   unsigned int : 23;
68 } jmmOptionalSupport;
______unchanged_portion_omitted_

194 typedef struct {
195   const char* name;               /* Name of the diagnostic command */
196   const char* description;        /* Short description */
197   const char* impact;             /* Impact on the JVM */
198   const char* permission_class;   /* Class name of the required permission if a
199   const char* permission_name;    /* Permission name of the required permission
200   const char* permission_action;  /* Action name of the required permission if
201   int         num_arguments;      /* Number of supported options or arguments *
202   jboolean    enabled;            /* True if the diagnostic command can be invo
203
193   const char* name;
194   const char* description;
195   const char* impact;
196   int         num_arguments;
197   jboolean    enabled;
204 } dcmdInfo;

206 typedef struct {
207   const char* name;               /* Option/Argument name*/
208   const char* description;        /* Short description */
209   const char* type;               /* Type: STRING, BOOLEAN, etc. */
210   const char* default_string;     /* Default value in a parsable string */
211   jboolean    mandatory;          /* True if the option/argument is mandatory *
212   jboolean    option;             /* True if it is an option, false if it is an
213                                   /* (see diagnosticFramework.hpp for option/ar
214   jboolean    multiple;           /* True is the option can be specified severa
215   int         position;           /* Expected position for this argument (this
216                                   /* meaningless for options) */
201   const char* name;
202   const char* description;
203   const char* type;
204   const char* default_string;
205   jboolean    mandatory;
206   jboolean    option;
207   int         position;
217 } dcmdArgInfo;

219 typedef struct jmmInterface_1_ {
220   void*       reserved1;
221   void*       reserved2;

223   jint        (JNICALL *GetVersion)           (JNIEnv *env);

225   jint        (JNICALL *GetOptionalSupport)   (JNIEnv *env,
226                                                jmmOptionalSupport* support_pt

228   /* This is used by JDK 6 and earlier.
229    * For JDK 7 and after, use GetInputArgumentArray.
230    */
231   jobject     (JNICALL *GetInputArguments)    (JNIEnv *env);

233   jint        (JNICALL *GetThreadInfo)        (JNIEnv *env,
234                                                jlongArray ids,
235                                                jint maxDepth,
236                                                jobjectArray infoArray);
```

```
237   jobjectArray (JNICALL *GetInputArgumentArray)  (JNIEnv *env);

239   jobjectArray (JNICALL *GetMemoryPools)         (JNIEnv* env, jobject mgr);

241   jobjectArray (JNICALL *GetMemoryManagers)      (JNIEnv* env, jobject pool);

243   jobject      (JNICALL *GetMemoryPoolUsage)     (JNIEnv* env, jobject pool);
244   jobject      (JNICALL *GetPeakMemoryPoolUsage) (JNIEnv* env, jobject pool);

246   void         (JNICALL *GetThreadAllocatedMemory)
247                                                  (JNIEnv *env,
248                                                   jlongArray ids,
249                                                   jlongArray sizeArray);

251   jobject      (JNICALL *GetMemoryUsage)         (JNIEnv* env, jboolean heap);

253   jlong        (JNICALL *GetLongAttribute)       (JNIEnv *env, jobject obj, jmmL
254   jboolean     (JNICALL *GetBoolAttribute)       (JNIEnv *env, jmmBoolAttribute
255   jboolean     (JNICALL *SetBoolAttribute)       (JNIEnv *env, jmmBoolAttribute

257   jint         (JNICALL *GetLongAttributes)      (JNIEnv *env,
258                                                   jobject obj,
259                                                   jmmLongAttribute* atts,
260                                                   jint count,
261                                                   jlong* result);

263   jobjectArray (JNICALL *FindCircularBlockedThreads) (JNIEnv *env);

265   // Not used in JDK 6 or JDK 7
266   jlong        (JNICALL *GetThreadCpuTime)       (JNIEnv *env, jlong thread_id);

268   jobjectArray (JNICALL *GetVMGlobalNames)       (JNIEnv *env);
269   jint         (JNICALL *GetVMGlobals)           (JNIEnv *env,
270                                                   jobjectArray names,
271                                                   jmmVMGlobal *globals,
272                                                   jint count);

274   jint         (JNICALL *GetInternalThreadTimes) (JNIEnv *env,
275                                                   jobjectArray names,
276                                                   jlongArray times);

278   jboolean     (JNICALL *ResetStatistic)         (JNIEnv *env,
279                                                   jvalue obj,
280                                                   jmmStatisticType type);

282   void         (JNICALL *SetPoolSensor)          (JNIEnv *env,
283                                                   jobject pool,
284                                                   jmmThresholdType type,
285                                                   jobject sensor);

287   jlong        (JNICALL *SetPoolThreshold)       (JNIEnv *env,
288                                                   jobject pool,
289                                                   jmmThresholdType type,
290                                                   jlong threshold);
291   jobject      (JNICALL *GetPoolCollectionUsage) (JNIEnv* env, jobject pool);

293   jint         (JNICALL *GetGCExtAttributeInfo)  (JNIEnv *env,
294                                                   jobject mgr,
295                                                   jmmExtAttributeInfo *ext_info,
296                                                   jint count);
297   void         (JNICALL *GetLastGCStat)          (JNIEnv *env,
298                                                   jobject mgr,
299                                                   jmmGCStat *gc_stat);

301   jlong        (JNICALL *GetThreadCpuTimeWithKind)
302                                                  (JNIEnv *env,
```



```
303                                                   jlong thread_id,
304                                                   jboolean user_sys_cpu_time);
305   void         (JNICALL *GetThreadCpuTimesWithKind)
306                                                  (JNIEnv *env,
307                                                   jlongArray ids,
308                                                   jlongArray timeArray,
309                                                   jboolean user_sys_cpu_time);

311   jint         (JNICALL *DumpHeap0)              (JNIEnv *env,
312                                                   jstring outputfile,
313                                                   jboolean live);
314   jobjectArray (JNICALL *FindDeadlocks)          (JNIEnv *env,
315                                                   jboolean object_monitors_only)
316   void         (JNICALL *SetVMGlobal)            (JNIEnv *env,
317                                                   jstring flag_name,
318                                                   jvalue  new_value);
319   void*        reserved6;
320   jobjectArray (JNICALL *DumpThreads)            (JNIEnv *env,
321                                                   jlongArray ids,
322                                                   jboolean lockedMonitors,
323                                                   jboolean lockedSynchronizers);
324   void         (JNICALL *SetGCNotificationEnabled) (JNIEnv *env,
325                                                     jobject mgr,
326                                                     jboolean enabled);
327   jobjectArray (JNICALL *GetDiagnosticCommands)  (JNIEnv *env);
328   void         (JNICALL *GetDiagnosticCommandInfo)
329                                                  (JNIEnv *env,
330                                                   jobjectArray cmds,
331                                                   dcmdInfo *infoArray);
332   void         (JNICALL *GetDiagnosticCommandArgumentsInfo)
333                                                  (JNIEnv *env,
334                                                   jstring commandName,
335                                                   dcmdArgInfo *infoArray);
336   jstring      (JNICALL *ExecuteDiagnosticCommand)
337                                                  (JNIEnv *env,
338                                                   jstring command);
339   void         (JNICALL *SetDiagnosticFrameworkNotificationEnabled)
340                                                  (JNIEnv *env,
341                                                   jboolean enabled);
342 } JmmInterface;
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   10587 Fri May  3 09:28:05 2013
new/src/share/native/sun/management/VMManagementImpl.c
**********************************************************
```

```
 1 /*
 2  * Copyright (c) 2003, 2013, Oracle and/or its affiliates. All rights reserved.
 2  * Copyright (c) 2003, 2011, Oracle and/or its affiliates. All rights reserved.
 3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 4  *
 5  * This code is free software; you can redistribute it and/or modify it
 6  * under the terms of the GNU General Public License version 2 only, as
 7  * published by the Free Software Foundation.  Oracle designates this
 8  * particular file as subject to the "Classpath" exception as provided
 9  * by Oracle in the LICENSE file that accompanied this code.
10  *
11  * This code is distributed in the hope that it will be useful, but WITHOUT
12  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
13  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
14  * version 2 for more details (a copy is included in the LICENSE file that
15  * accompanied this code).
16  *
17  * You should have received a copy of the GNU General Public License version
18  * 2 along with this work; if not, write to the Free Software Foundation,
19  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
20  *
21  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
22  * or visit www.oracle.com if you need additional information or have any
23  * questions.
24  */

26 #include <jni.h>
27 #include <stdlib.h>
28 #include "jvm.h"
29 #include "management.h"
30 #include "sun_management_VMManagementImpl.h"

32 #define MAX_VERSION_LEN   20

34 JNIEXPORT jstring JNICALL
35 Java_sun_management_VMManagementImpl_getVersion0
36   (JNIEnv *env, jclass dummy)
37 {
38     char buf[MAX_VERSION_LEN];
39     jstring version_string = NULL;

41     unsigned int major = ((unsigned int) jmm_version & 0x0FFF0000) >> 16;
42     unsigned int minor = ((unsigned int) jmm_version & 0xFF00) >> 8;

44     // for internal use
45     unsigned int micro = (unsigned int) jmm_version & 0xFF;

47     sprintf(buf, "%d.%d", major, minor);
48     version_string = (*env)->NewStringUTF(env, buf);
49     return version_string;
50 }
```

*unchanged_portion_omitted_*

```
62 JNIEXPORT void JNICALL
63 Java_sun_management_VMManagementImpl_initOptionalSupportFields
64   (JNIEnv *env, jclass cls)
65 {
66     jmmOptionalSupport mos;
67     jint ret = jmm_interface->GetOptionalSupport(env, &mos);

69     jboolean value;
```



```
71     value = mos.isCompilationTimeMonitoringSupported;
72     setStaticBooleanField(env, cls, "compTimeMonitoringSupport", value);

74     value = mos.isThreadContentionMonitoringSupported;
75     setStaticBooleanField(env, cls, "threadContentionMonitoringSupport", value);

77     value = mos.isCurrentThreadCpuTimeSupported;
78     setStaticBooleanField(env, cls, "currentThreadCpuTimeSupport", value);

80     value = mos.isOtherThreadCpuTimeSupported;
81     setStaticBooleanField(env, cls, "otherThreadCpuTimeSupport", value);

83     value = mos.isBootClassPathSupported;
84     setStaticBooleanField(env, cls, "bootClassPathSupport", value);

86     if (jmm_version >= JMM_VERSION_1_1) {
87         value = mos.isObjectMonitorUsageSupported;
88         setStaticBooleanField(env, cls, "objectMonitorUsageSupport", value);

90         value = mos.isSynchronizerUsageSupported;
91         setStaticBooleanField(env, cls, "synchronizerUsageSupport", value);
92     } else {
93         setStaticBooleanField(env, cls, "objectMonitorUsageSupport", JNI_FALSE);
94         setStaticBooleanField(env, cls, "synchronizerUsageSupport", JNI_FALSE);
95     }

97     value = mos.isThreadAllocatedMemorySupported;
98     setStaticBooleanField(env, cls, "threadAllocatedMemorySupport", value);

100    value = mos.isRemoteDiagnosticCommandsSupported;
101    setStaticBooleanField(env, cls, "remoteDiagnosticCommandsSupport", value);

103    if ((jmm_version > JMM_VERSION_1_2) ||
104        (jmm_version == JMM_VERSION_1_2 && ((jmm_version&0xFF) >= 1))) {
105        setStaticBooleanField(env, cls, "gcNotificationSupport", JNI_TRUE);
106    } else {
107        setStaticBooleanField(env, cls, "gcNotificationSupport", JNI_FALSE);
108    }
109 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    8257 Fri May  3 09:28:06 2013
new/test/java/lang/management/MXBean/MXBeanBehavior.java
**********************************************************
```

```
/*
 * Copyright (c) 2005, 2013 Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2005, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.
 *
 * This code is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 * or visit www.oracle.com if you need additional information or have any
 * questions.
 */

/*
 * @test
 * @bug     6320211
 * @summary Check that java.lang.management MXBeans have the same behavior
 *          as user MXBeans
 * @author  Eamonn McManus
 * @run     main/othervm MXBeanBehavior
 */

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

public class MXBeanBehavior {
    // Exclude list: list of platform MBeans that are not MXBeans
    public static final HashSet<String> excludeList = new HashSet<>(
            Arrays.asList("com.sun.management:type=DiagnosticCommand"));

    public static void main(String[] args) throws Exception {
        MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();

        /* Test that all the MBeans in the java.* and com.sun.management*
           domains are MXBeans with the appropriate behavior.  */
        Set<ObjectName> names = mbs.queryNames(new ObjectName("java.*:*"),
                                               null);
        names.addAll(mbs.queryNames(new ObjectName("com.sun.management*:*"),
                                    null));
        for (ObjectName name : names)
            test(mbs, name);

        /* Now do some rudimentary testing of inter-MXBean references.
           It should be possible for a user MXBean to return e.g.  the
           CompilationMXBean from the platform from an attribute of
           type CompilationMXBean, and have the MXBean infrastructure
           map this into that MXBean's standard ObjectName.  It should
           also be possible for a proxy for this user MXBean to have
           this attribute's value mapped back into a CompilationMXBean
```



```
           instance, which however will be another proxy rather than
           the original object.  Finally, it should be possible to set
           the attribute in the user's MXBean through a proxy, giving
           the real CompilationMXBean as an argument, and have this be
           translated into that MXBean's standard ObjectName.  The
           user's MXBean will receive a proxy in this case, though we
           don't check that.  */
        ObjectName refName = new ObjectName("d:type=CompilationRef");
        mbs.registerMBean(new CompilationImpl(), refName);
        CompilationRefMXBean refProxy =
            JMX.newMXBeanProxy(mbs, refName, CompilationRefMXBean.class);
        refProxy.getCompilationMXBean();
        refProxy.setCompilationMXBean(ManagementFactory.getCompilationMXBean());
        ObjectName on =
            (ObjectName) mbs.getAttribute(refName, "CompilationMXBean");
        checkEqual(on, new ObjectName(ManagementFactory.COMPILATION_MXBEAN_NAME)
                   "Referenced object name");
        mbs.setAttribute(refName, new Attribute("CompilationMXBean", on));

        System.out.println("TEST PASSED");
    }

    /* Check the behavior of this MXBean to ensure that it conforms to
       what is expected of all MXBeans as detailed in
       javax.management.MXBean.  Its MBeanInfo should have a
       Descriptor with the fields mxbean and interfaceClassName, and
       furthermore we know that our implementation sets immutableInfo
       here.  Each attribute should have Descriptor with the fields
       openType and originalType that have appropriate values.  We
       don't currently check operations though the same considerations
       would apply there.  (If the MBeanInfo and MBeanAttributeInfo
       tests pass we can reasonably suppose that this MXBean will
       behave the same as all other MXBeans, so MBeanOperationInfo,
       MBeanNotificationInfo, and MBeanConstructorInfo will be covered
       by generic MXBean tests.
     */
    private static void test(MBeanServer mbs, ObjectName name) throws Exception
        if(excludeList.contains(name.getCanonicalName())) {
            // Skipping not MXBean objects.
            return;
        }
        System.out.println("Testing: " + name);

        MBeanInfo mbi = mbs.getMBeanInfo(name);
        Descriptor mbid = mbi.getDescriptor();
        Object[] values = mbid.getFieldValues("immutableInfo",
                                              "interfaceClassName",
                                              "mxbean");
        checkEqual(values[0], "true", name + " immutableInfo field");
        checkEqual(values[2], "true", name + " mxbean field");
        String interfaceClassName = (String) values[1];
        if (!mbs.isInstanceOf(name, interfaceClassName)) {
            throw new RuntimeException(name + " not instance of " +
                                       interfaceClassName);
        }
        Class interfaceClass = Class.forName(interfaceClassName);
        for (MBeanAttributeInfo mbai : mbi.getAttributes()) {
            Descriptor mbaid = mbai.getDescriptor();
            Object[] avalues = mbaid.getFieldValues("openType",
                                                    "originalType");
            if (avalues[0] == null || avalues[1] == null) {
                throw new RuntimeException("Null attribute descriptor fields: "
                                           Arrays.toString(avalues));
            }
            if (mbai.isReadable()) {
                String mname = (mbai.isIs() ? "is" : "get") + mbai.getName();
```

```
128                 Method m = interfaceClass.getMethod(mname);
129                 Type t = m.getGenericReturnType();
130                 String ret =
131                     (t instanceof Class) ? ((Class) t).getName() : t.toString();
132                 if (!ret.equals(avalues[1])) {
133                     final String msg =
134                         name + " attribute " + mbai.getName() + " has wrong " +
135                         "originalType: " + avalues[1] + " vs " + ret;
136                     throw new RuntimeException(msg);
137                 }
138             }
139         }
140     }

142     private static void checkEqual(Object x, Object y, String what) {
143         final boolean eq;
144         if (x == y)
145             eq = true;
146         else if (x == null)
147             eq = false;
148         else
149             eq = x.equals(y);
150         if (!eq)
151             throw new RuntimeException(what + " should be " + y + ", is " + x);
152     }

154     public static interface CompilationRefMXBean {
155         public CompilationMXBean getCompilationMXBean();
156         public void setCompilationMXBean(CompilationMXBean mxb);
157     }

159     public static class CompilationImpl implements CompilationRefMXBean {
160         public CompilationMXBean getCompilationMXBean() {
161             return ManagementFactory.getCompilationMXBean();
162         }

164         public void setCompilationMXBean(CompilationMXBean mxb) {
165             if (mxb == ManagementFactory.getCompilationMXBean())
166                 return;
167             MBeanServerInvocationHandler mbsih = (MBeanServerInvocationHandler)
168                 Proxy.getInvocationHandler(mxb);
169             ObjectName expectedName;
170             try {
171                 expectedName =
172                     new ObjectName(ManagementFactory.COMPILATION_MXBEAN_NAME);
173             } catch (MalformedObjectNameException e) {
174                 throw new RuntimeException(e);
175             }
176             checkEqual(mbsih.getObjectName(), expectedName,
177                        "Proxy name in setCompilationMXBean");
178         }
179     }
180 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    6226 Fri May  3 09:28:07 2013
new/test/java/lang/management/ManagementFactory/MBeanServerMXBeanUnsupportedTest
.java
**********************************************************
```

```
 1 /*
 2  * Copyright (c) 2006, 2013 Oracle and/or its affiliates. All rights reserved.
 2  * Copyright (c) 2006, 2011, Oracle and/or its affiliates. All rights reserved.
 3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 4  *
 5  * This code is free software; you can redistribute it and/or modify it
 6  * under the terms of the GNU General Public License version 2 only, as
 7  * published by the Free Software Foundation.
 8  *
 9  * This code is distributed in the hope that it will be useful, but WITHOUT
10  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
11  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
12  * version 2 for more details (a copy is included in the LICENSE file that
13  * accompanied this code).
14  *
15  * You should have received a copy of the GNU General Public License version
16  * 2 along with this work; if not, write to the Free Software Foundation,
17  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
18  *
19  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
20  * or visit www.oracle.com if you need additional information or have any
21  * questions.
22  */

24 /*
25  * @test
26  * @bug 6403794
27  * @summary Test that all the platform MXBeans are wrapped in StandardMBean so
28  *          an MBeanServer which does not have support for MXBeans can be used.
29  * @author Luis-Miguel Alventosa
30  * @run clean MBeanServerMXBeanUnsupportedTest
31  * @run build MBeanServerMXBeanUnsupportedTest
32  * @run main/othervm MBeanServerMXBeanUnsupportedTest
33  */

35 import java.lang.management.ManagementFactory;
36 import java.lang.reflect.InvocationHandler;
37 import java.lang.reflect.Method;
38 import java.lang.reflect.Proxy;
39 import java.util.Arrays;
40 import java.util.HashSet;
41 import javax.management.MBeanServer;
42 import javax.management.MBeanServerBuilder;
43 import javax.management.MBeanServerDelegate;
44 import javax.management.ObjectName;
45 import javax.management.StandardMBean;
46 import javax.management.remote.MBeanServerForwarder;

48 public class MBeanServerMXBeanUnsupportedTest {

50     /**
51      * An MBeanServerBuilder that returns an MBeanServer which throws a
52      * RuntimeException if MXBeans are not converted into StandardMBean.
53      */
54     public static class MBeanServerBuilderImpl extends MBeanServerBuilder {

56         private final MBeanServerBuilder inner;

58         public MBeanServerBuilderImpl() {
59             inner = new MBeanServerBuilder();
60         }
```



```
62         public MBeanServer newMBeanServer(
63                 String defaultDomain,
64                 MBeanServer outer,
65                 MBeanServerDelegate delegate) {
66             final MBeanServerForwarder mbsf =
67                     MBeanServerForwarderInvocationHandler.newProxyInstance();

69             final MBeanServer innerMBeanServer =
70                     inner.newMBeanServer(defaultDomain,
71                     (outer == null ? mbsf : outer),
72                     delegate);

74             mbsf.setMBeanServer(innerMBeanServer);
75             return mbsf;
76         }
77     }

79     /**
80      * An MBeanServerForwarderInvocationHandler that throws a
81      * RuntimeException if we try to register a non StandardMBean.
82      */
83     public static class MBeanServerForwarderInvocationHandler
84             implements InvocationHandler {

86         public static final HashSet<String> excludeList = new HashSet<String>(
87             Arrays.asList("com.sun.management:type=DiagnosticCommand"));

89         public static MBeanServerForwarder newProxyInstance() {

91             final InvocationHandler handler =
92                     new MBeanServerForwarderInvocationHandler();

94             final Class[] interfaces =
95                     new Class[] {MBeanServerForwarder.class};

97             Object proxy = Proxy.newProxyInstance(
98                     MBeanServerForwarder.class.getClassLoader(),
99                     interfaces,
100                    handler);

102            return MBeanServerForwarder.class.cast(proxy);
103        }

105        public Object invoke(Object proxy, Method method, Object[] args)
106                throws Throwable {

108            final String methodName = method.getName();

110            if (methodName.equals("getMBeanServer")) {
111                return mbs;
112            }

114            if (methodName.equals("setMBeanServer")) {
115                if (args[0] == null)
116                    throw new IllegalArgumentException("Null MBeanServer");
117                if (mbs != null)
118                    throw new IllegalArgumentException("MBeanServer object " +
119                            "already initialized");
120                mbs = (MBeanServer) args[0];
121                return null;
122            }

124            if (methodName.equals("registerMBean")) {
125                Object mbean = args[0];
126                ObjectName name = (ObjectName) args[1];
```

```
127                 String domain = name.getDomain();
128                 System.out.println("registerMBean: class=" +
129                         mbean.getClass().getName() + "\tname=" + name);
130                 Object result = method.invoke(mbs, args);
131                 if (domain.equals("java.lang") ||
132                     domain.equals("java.util.logging") ||
133                     domain.equals("com.sun.management")) {
134                     if(!excludeList.contains(name.getCanonicalName())) {
135                         String mxbean = (String)
136                             mbs.getMBeanInfo(name).getDescriptor().getFieldValue
137                         if (mxbean == null || !mxbean.equals("true")) {
138                             throw new RuntimeException(
139                                 "Platform MBeans must be MXBeans!");
140                         }
141                         if (!(mbean instanceof StandardMBean)) {
142                             throw new RuntimeException(
143                                 "MXBeans must be wrapped in StandardMBean!");
144                         }
145                     }
146                 }
147                 return result;
148             }

150             return method.invoke(mbs, args);
151         }

153         private MBeanServer mbs;
154     }

156     /*
157      * Standalone entry point.
158      *
159      * Run the test and report to stdout.
160      */
161     public static void main(String args[]) throws Exception {
162         System.setProperty("javax.management.builder.initial",
163                 MBeanServerBuilderImpl.class.getName());
164         try {
165             ManagementFactory.getPlatformMBeanServer();
166         } catch (RuntimeException e) {
167             System.out.println(">>> Unhappy Bye, Bye!");
168             throw e;
169         }
170         System.out.println(">>> Happy Bye, Bye!");
171     }
172 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    9450 Fri May  3 09:28:09 2013
new/src/share/classes/com/sun/management/DiagnosticCommandMBean.java
**********************************************************
```

```
/*
 * Copyright (c) 2013, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Oracle designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Oracle in the LICENSE file that accompanied this code.
 *
 * This code is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 * or visit www.oracle.com if you need additional information or have any
 * questions.
 */

package com.sun.management;

import java.lang.management.PlatformManagedObject;
import javax.management.DynamicMBean;

/**
 * Management interface for the diagnostic commands for the HotSpot Virtual Mach
 *
 * <p>The {code DiagnosticCommandMBean} is registered to the
 * {@linkplain java.lang.management.ManagementFactory#getPlatformMBeanServer
 * platform MBeanServer} as are other platform MBeans.
 *
 * <p>The {@link javax.management.ObjectName ObjectName} for uniquely identifyin
 * the diagnostic MBean within an MBeanServer is:
 * <blockquote>
 *    {@code com.sun.management:type=DiagnosticCommand}
 * </blockquote>
 *
 * <p>This MBean is a {@link javax.management.DynamicMBean DynamicMBean}
 * and also a {@link javax.management.NotificationEmitter}.
 * The {@code DiagnosticCommandMBean} is generated at runtime and is subject to
 * modifications during the lifetime of the Java virtual machine.
 *
 * A <em>diagnostic command</em> is represented as an operation of
 * the {@code DiagnosticCommandMBean} interface. Each diagnostic command has:
 * <ul>
 * <li>the diagnostic command name which is the name being referenced in
 *     the HotSpot Virtual Machine</li>
 * <li>the MBean operation name which is the
 *     {@linkplain javax.management.MBeanOperationInfo#getName() name}
 *     generated for the diagnostic command operation invocation.
 *     The MBean operation name is implementation dependent</li>
 * </ul>
 *
 * The recommended way to transform a diagnostic command name into a MBean
 * operation name is as follows:
 * <ul>
```



```
 * <li>All characters from the first one to the first dot are set to be
 *     lower-case characters</li>
 * <li>Every dot or underline character is removed and the following
 *   character is set to be an upper-case character</li>
 * <li>All other characters are copied without modification</li>
 * </ul>
 *
 * <p>The diagnostic command name is always provided with the meta-data on the
 * operation in a field named {@code dcmd.name} (see below).
 *
 * <p>A diagnostic command may or may not support options or arguments.
 * All the operations return {@code String} and either take
 * no parameter for operations that do not support any option or argument,
 * or take a {@code String[]} parameter for operations that support at least
 * one option or argument.
 * Each option or argument must be stored in a single String.
 * Options or arguments split across several String instances are not supported.
 *
 * <p>The distinction between options and arguments: options are identified by
 * the option name while arguments are identified by their position in the
 * command line. Options and arguments are processed in the order of the array
 * passed to the invocation method.
 *
 * <p>Like any operation of a dynamic MBean, each of these operations is
 * described by {@link javax.management.MBeanOperationInfo MBeanOperationInfo}
 * instance. Here's the values returned by this object:
 * <ul>
 *  <li>{@link javax.management.MBeanOperationInfo#getName() getName()}
 *      returns the operation name generated from the diagnostic command name</l
 *  <li>{@link javax.management.MBeanOperationInfo#getDescription() getDescripti
 *      returns the diagnostic command description
 *      (the same as the one return in the 'help' command)</li>
 *  <li>{@link javax.management.MBeanOperationInfo#getImpact() getImpact()}
 *      returns <code>ACTION_INFO</code></li>
 *  <li>{@link javax.management.MBeanOperationInfo#getReturnType() getReturnType
 *      returns {@code java.lang.String}</li>
 *  <li>{@link javax.management.MBeanOperationInfo#getDescriptor() getDescriptor
 *      returns a Descriptor instance (see below)</li>
 * </ul>
 *
 * <p>The {@link javax.management.Descriptor Descriptor}
 * is a collection of fields containing additional
 * meta-data for a JMX element. A field is a name and an associated value.
 * The additional meta-data provided for an operation associated with a
 * diagnostic command are described in the table below:
 * <p>
 *
 * <table border="1" cellpadding="5">
 *   <tr>
 *     <th>Name</th><th>Type</th><th>Description</th>
 *   </tr>
 *   <tr>
 *     <td>dcmd.name</td><td>String</td>
 *     <td>The original diagnostic command name (not the operation name)</td>
 *   </tr>
 *   <tr>
 *     <td>dcmd.description</td><td>String</td>
 *     <td>The diagnostic command description</td>
 *   </tr>
 *   <tr>
 *     <td>dcmd.help</td><td>String</td>
 *     <td>The full help message for this diagnostic command (same output as
 *         the one produced by the 'help' command)</td>
 *   </tr>
 *   <tr>
 *     <td>dcmd.vmImpact</td><td>String</td>
```

```
129  *     <td>The impact of the diagnostic command,
130  *      this value is the same as the one printed in the 'impact'
131  *      section of the help message of the diagnostic command, and it
132  *      is different from the getImpact() of the MBeanOperationInfo</td>
133  *   </tr>
134  *   <tr>
135  *     <td>dcmd.enabled</td><td>boolean</td>
136  *     <td>True if the diagnostic command is enabled, false otherwise</td>
137  *   </tr>
138  *   <tr>
139  *     <td>dcmd.permissionClass</td><td>String</td>
140  *     <td>Some diagnostic command might require a specific permission to be
141  *          executed, in addition to the MBeanPermission to invoke their
142  *          associated MBean operation. This field returns the fully qualified
143  *          name of the permission class or null if no permission is required
144  *   </td>
145  *   </tr>
146  *   <tr>
147  *     <td>dcmd.permissionName</td><td>String</td>
148  *     <td>The fist argument of the permission required to execute this
149  *          diagnostic command or null if no permission is required</td>
150  *   </tr>
151  *   <tr>
152  *     <td>dcmd.permissionAction</td><td>String</td>
153  *     <td>The second argument of the permission required to execute this
154  *          diagnostic command or null if the permission constructor has only
155  *          one argument (like the ManagementPermission) or if no permission
156  *          is required</td>
157  *   </tr>
158  *   <tr>
159  *     <td>dcmd.arguments</td><td>Descriptor</td>
160  *     <td>A Descriptor instance containing the descriptions of options and
161  *          arguments supported by the diagnostic command (see below)</td>
162  *   </tr>
163  * </table>
164  * <p>
165  *
166  * <p>The description of parameters (options or arguments) of a diagnostic
167  * command is provided within a Descriptor instance. In this Descriptor,
168  * each field name is a parameter name, and each field value is itself
169  * a Descriptor instance. The fields provided in this second Descriptor
170  * instance are described in the table below:
171  *
172  * <table border="1" cellpadding="5">
173  *   <tr>
174  *     <th>Name</th><th>Type</th><th>Description</th>
175  *   </tr>
176  *   <tr>
177  *     <td>dcmd.arg.name</td><td>String</td>
178  *     <td>The name of the parameter</td>
179  *   </tr>
180  *   <tr>
181  *     <td>dcmd.arg.type</td><td>String</td>
182  *     <td>The type of the parameter. The returned String is the name of a type
183  *          recognized by the diagnostic command parser. These types are not
184  *          Java types and are implementation dependent.
185  *          </td>
186  *   </tr>
187  *   <tr>
188  *     <td>dcmd.arg.description</td><td>String</td>
189  *     <td>The parameter description</td>
190  *   </tr>
191  *   <tr>
192  *     <td>dcmd.arg.isMandatory</td><td>boolean</td>
193  *     <td>True if the parameter is mandatory, false otherwise</td>
194  *   </tr>
```



```
195  *   <tr>
196  *     <td>dcmd.arg.isOption</td><td>boolean</td>
197  *     <td>True if the parameter is an option, false if it is an argument</td>
198  *   </tr>
199  *   <tr>
200  *     <td>dcmd.arg.isMultiple</td><td>boolean</td>
201  *     <td>True if the parameter can be specified several times, false
202  *          otherwise</td>
203  *   </tr>
204  * </table>
205  *
206  * <p>When the set of diagnostic commands currently supported by the Java
207  * Virtual Machine is modified, the {@code DiagnosticCommandMBean} emits
208  * a {@link Notification} with a {@linkplain Notification#getType() type} of
209  * <a href="{@docRoot}/../../../../api/javax/management/MBeanInfo.html#info-chan
210  * {@code "jmx.mbean.info.changed"}</a> and a {@linkplain
211  * Notification#getUserData() userData} that is the new {@code MBeanInfo}.
212  *
213  * @since 8
214  */
215 public interface DiagnosticCommandMBean extends DynamicMBean
216 {

218 }
```

```
**********************************************************
    5268 Fri May  3 09:28:10 2013
new/src/share/classes/sun/management/DiagnosticCommandArgumentInfo.java
**********************************************************
```

```
  1 /*
  2  * Copyright (c) 2013, Oracle and/or its affiliates. All rights reserved.
  3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
  4  *
  5  * This code is free software; you can redistribute it and/or modify it
  6  * under the terms of the GNU General Public License version 2 only, as
  7  * published by the Free Software Foundation.  Oracle designates this
  8  * particular file as subject to the "Classpath" exception as provided
  9  * by Oracle in the LICENSE file that accompanied this code.
 10  *
 11  * This code is distributed in the hope that it will be useful, but WITHOUT
 12  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 13  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 14  * version 2 for more details (a copy is included in the LICENSE file that
 15  * accompanied this code).
 16  *
 17  * You should have received a copy of the GNU General Public License version
 18  * 2 along with this work; if not, write to the Free Software Foundation,
 19  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 20  *
 21  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 22  * or visit www.oracle.com if you need additional information or have any
 23  * questions.
 24  */

 26 package sun.management;

 28 /**
 29  * Diagnostic Command Argument information. It contains the description
 30  * of one parameter of the diagnostic command. A parameter can either be an
 31  * option or an argument. Options are identified by the option name while
 32  * arguments are identified by their position in the command line. The generic
 33  * syntax of a diagnostic command is:
 34  *  <blockquote>
 35  *    &lt;command name&gt; [&lt;option&gt;=&lt;value&gt;] [&lt;argument_value&gt
 36  * </blockquote>
 37  * Example:
 38  * <blockquote>
 39  * command_name option1=value1 option2=value argumentA argumentB argumentC
 40  * </blockquote>
 41  * In this command line, the diagnostic command receives five parameters, two
 42  * options named {@code option1} and {@code option2}, and three arguments.
 43  * argumentA's position is 0, argumentB's position is 1 and argumentC's
 44  * position is 2.
 45  *
 46  * @since 8
 47  */

 49 class DiagnosticCommandArgumentInfo {
 50     private final String name;
 51     private final String description;
 52     private final String type;
 53     private final String defaultValue;
 54     private final boolean mandatory;
 55     private final boolean option;
 56     private final boolean multiple;
 57     private final int position;

 59     /**
 60      * Returns the argument name.
 61      *
 62      * @return the argument name
```



```
 63      */
 64     String getName() {
 65         return name;
 66     }

 68     /**
 69      * Returns the argument description.
 70      *
 71      * @return the argument description
 72      */
 73     String getDescription() {
 74         return description;
 75     }

 77     /**
 78      * Returns the argument type.
 79      *
 80      * @return the argument type
 81      */
 82     String getType() {
 83         return type;
 84     }

 86     /**
 87      * Returns the default value as a String if a default value
 88      * is defined, null otherwise.
 89      *
 90      * @return the default value as a String if a default value
 91      * is defined, null otherwise.
 92      */
 93     String getDefault() {
 94         return defaultValue;
 95     }

 97     /**
 98      * Returns {@code true} if the argument is mandatory,
 99      *         {@code false} otherwise.
100      *
101      * @return {@code true} if the argument is mandatory,
102      *         {@code false} otherwise
103      */
104     boolean isMandatory() {
105         return mandatory;
106     }

108     /**
109      * Returns {@code true} if the argument is an option,
110      *         {@code false} otherwise. Options have to be specified using the
111      *         &lt;key&gt;=&lt;value&gt; syntax on the command line, while other
112      *         arguments are specified with a single &lt;value&gt; field and are
113      *         identified by their position on command line.
114      *
115      * @return {@code true} if the argument is an option,
116      *         {@code false} otherwise
117      */
118     boolean isOption() {
119         return option;
120     }

122     /**
123      * Returns {@code true} if the argument can be specified multiple times,
124      *         {@code false} otherwise.
125      *
126      * @return {@code true} if the argument can be specified multiple times,
127      *         {@code false} otherwise
128      */
```

```
129     boolean isMultiple() {
130         return multiple;
131     }

133     /**
134      * Returns the expected position of this argument if it is not an option,
135      *         -1 otherwise. Argument position if defined from left to right,
136      *         starting at zero and ignoring the diagnostic command name and
137      *         options.
138      *
139      * @return the expected position of this argument if it is not an option,
140      *         -1 otherwise.
141      */
142     int getPosition() {
143         return position;
144     }

146     DiagnosticCommandArgumentInfo(String name, String description,
147                                   String type, String defaultValue,
148                                   boolean mandatory, boolean option,
149                                   boolean multiple, int position) {
150         this.name = name;
151         this.description = description;
152         this.type = type;
153         this.defaultValue = defaultValue;
154         this.mandatory = mandatory;
155         this.option = option;
156         this.multiple = multiple;
157         this.position = position;
158     }
159 }
```

```
*********************************************************
   15399 Fri May  3 09:28:11 2013
new/src/share/classes/sun/management/DiagnosticCommandImpl.java
*********************************************************
```

```
/*
 * Copyright (c) 2013, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Oracle designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Oracle in the LICENSE file that accompanied this code.
 *
 * This code is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 * or visit www.oracle.com if you need additional information or have any
 * questions.
 */

package sun.management;

import com.sun.management.DiagnosticCommandMBean;
import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;
import java.security.Permission;
import java.util.*;
import javax.management.*;

/**
 * Implementation class for the diagnostic commands subsystem.
 *
 * @since 8
 */
class DiagnosticCommandImpl extends NotificationEmitterSupport
    implements DiagnosticCommandMBean {

    private final VMManagement jvm;
    private volatile Map<String, Wrapper> wrappers = null;
    private static final String strClassName = "".getClass().getName();
    private static final String strArrayClassName = String[].class.getName();
    private final boolean isSupported;

    @Override
    public Object getAttribute(String attribute) throws AttributeNotFoundExcepti
        MBeanException, ReflectionException {
        throw new AttributeNotFoundException(attribute);
    }

    @Override
    public void setAttribute(Attribute attribute) throws AttributeNotFoundExcept
        InvalidAttributeValueException, MBeanException, ReflectionException {
        throw new AttributeNotFoundException(attribute.getName());
    }

    @Override
    public AttributeList getAttributes(String[] attributes) {
```



```
        throw new UnsupportedOperationException("Not supported.");
    }

    @Override
    public AttributeList setAttributes(AttributeList attributes) {
        throw new UnsupportedOperationException("Not supported.");
    }

    private class Wrapper {

        String name;
        String cmd;
        DiagnosticCommandInfo info;
        Permission permission;

        Wrapper(String name, String cmd, DiagnosticCommandInfo info)
                throws InstantiationException {
            this.name = name;
            this.cmd = cmd;
            this.info = info;
            this.permission = null;
            if (info.getPermissionClass() != null) {
                try {
                    Class c = Class.forName(info.getPermissionClass());
                    if (info.getPermissionAction() == null) {
                        try {
                            Constructor constructor = c.getConstructor(String.cl
                            permission = (Permission) constructor.newInstance(in

                        } catch (InstantiationException | IllegalAccessException
                                | IllegalArgumentException | InvocationTargetExc
                                | NoSuchMethodException | SecurityException ex)
                    }
                    if (permission == null) {
                        try {
                            Constructor constructor = c.getConstructor(String.cl
                            permission = (Permission) constructor.newInstance(
                                    info.getPermissionName(),
                                    info.getPermissionAction());
                        } catch (InstantiationException | IllegalAccessException
                                | IllegalArgumentException | InvocationTargetExc
                                | NoSuchMethodException | SecurityException ex)
                    }
                } catch (ClassNotFoundException ex) { }
                if (permission == null) {
                    throw new InstantiationException("Unable to instantiate requ
                }
            }
        }

        public String execute(String[] args) {
            if (permission != null) {
                SecurityManager sm = System.getSecurityManager();
                if (sm != null) {
                    sm.checkPermission(permission);
                }
            }
            if(args == null) {
                return executeDiagnosticCommand(cmd);
            } else {
                StringBuilder sb = new StringBuilder();
                sb.append(cmd);
                for(int i=0; i<args.length; i++) {
                    if(args[i] == null) {
                        throw new IllegalArgumentException("Invalid null argumen
                    }
```

```
129                     sb.append(" ");
130                     sb.append(args[i]);
131                 }
132                 return executeDiagnosticCommand(sb.toString());
133             }
134         }
135     }

137     DiagnosticCommandImpl(VMManagement jvm) {
138         this.jvm = jvm;
139         isSupported = jvm.isRemoteDiagnosticCommandsSupported();
140     }

142     private static class OperationInfoComparator implements Comparator<MBeanOper
143         @Override
144         public int compare(MBeanOperationInfo o1, MBeanOperationInfo o2) {
145             return o1.getName().compareTo(o2.getName());
146         }
147     }

149     @Override
150     public MBeanInfo getMBeanInfo() {
151         SortedSet<MBeanOperationInfo> operations = new TreeSet<>(new OperationIn
152         Map<String, Wrapper> wrappersmap;
153         if (!isSupported) {
154             wrappersmap = (Map<String, Wrapper>) Collections.EMPTY_MAP;
155         } else {
156             try {
157                 String[] command = getDiagnosticCommands();
158                 DiagnosticCommandInfo[] info = getDiagnosticCommandInfo(command)
159                 MBeanParameterInfo stringArgInfo[] = new MBeanParameterInfo[]{
160                     new MBeanParameterInfo("arguments", strArrayClassName,
161                     "Array of Diagnostic Commands Arguments and Options")
162                 };
163                 wrappersmap = new HashMap<>();
164                 for (int i = 0; i < command.length; i++) {
165                     String name = transform(command[i]);
166                     try {
167                         Wrapper w = new Wrapper(name, command[i], info[i]);
168                         wrappersmap.put(name, w);
169                         operations.add(new MBeanOperationInfo(
170                                 w.name,
171                                 w.info.getDescription(),
172                                 (w.info.getArgumentsInfo() == null
173                                     || w.info.getArgumentsInfo().isEmpty())
174                                     ? null : stringArgInfo,
175                                 strClassName,
176                                 MBeanOperationInfo.ACTION_INFO,
177                                 commandDescriptor(w)));
178                     } catch (InstantiationException ex) {
179                         // If for some reasons the creation of a diagnostic comm
180                         // wrappers fails, the diagnostic command is just ignore
181                         // and won't appear in the DynamicMBean
182                     }
183                 }
184             } catch (IllegalArgumentException | UnsupportedOperationException e)
185                 wrappersmap = (Map<String, Wrapper>) Collections.EMPTY_MAP;
186             }
187         }
188         wrappers =  Collections.unmodifiableMap(wrappersmap);
189         HashMap<String, Object> map = new HashMap<>();
190         map.put("immutableInfo", "false");
191         map.put("interfaceClassName","com.sun.management.DiagnosticCommandMBean"
192         map.put("mxbean", "false");
193         Descriptor desc = new ImmutableDescriptor(map);
194         return new MBeanInfo(
```



```
195                 this.getClass().getName(),
196                 "Diagnostic Commands",
197                 null, // attributes
198                 null, // constructors
199                 operations.toArray(new MBeanOperationInfo[operations.size()]), /
200                 getNotificationInfo(), // notifications
201                 desc);
202     }

204     @Override
205     public Object invoke(String actionName, Object[] params, String[] signature)
206             throws MBeanException, ReflectionException {
207         if (!isSupported) {
208             throw new UnsupportedOperationException();
209         }
210         if (wrappers == null) {
211             getMBeanInfo();
212         }
213         Wrapper w = wrappers.get(actionName);
214         if (w != null) {
215             if (w.info.getArgumentsInfo().isEmpty()
216                     && (params == null || params.length == 0)
217                     && (signature == null || signature.length == 0)) {
218                 return w.execute(null);
219             } else if((params != null && params.length == 1)
220                     && (signature != null && signature.length == 1
221                     && signature[0] != null
222                     && signature[0].compareTo(strArrayClassName) == 0)) {
223                 return w.execute((String[]) params[0]);
224             }
225         }
226         throw new ReflectionException(new NoSuchMethodException(actionName));
227     }

229     private static String transform(String name) {
230         StringBuilder sb = new StringBuilder();
231         boolean toLower = true;
232         boolean toUpper = false;
233         for (int i = 0; i < name.length(); i++) {
234             char c = name.charAt(i);
235             if (c == '.' || c == '_') {
236                 toLower = false;
237                 toUpper = true;
238             } else {
239                 if (toUpper) {
240                     toUpper = false;
241                     sb.append(Character.toUpperCase(c));
242                 } else if(toLower) {
243                     sb.append(Character.toLowerCase(c));
244                 } else {
245                     sb.append(c);
246                 }
247             }
248         }
249         return sb.toString();
250     }

252     private Descriptor commandDescriptor(Wrapper w) throws IllegalArgumentExcept
253         HashMap<String, Object> map = new HashMap<>();
254         map.put("dcmd.name", w.info.getName());
255         map.put("dcmd.description", w.info.getDescription());
256         map.put("dcmd.vmImpact", w.info.getImpact());
257         map.put("dcmd.permissionClass", w.info.getPermissionClass());
258         map.put("dcmd.permissionName", w.info.getPermissionName());
259         map.put("dcmd.permissionAction", w.info.getPermissionAction());
260         map.put("dcmd.enabled", w.info.isEnabled());
```

```
261         StringBuilder sb = new StringBuilder();
262         sb.append("help ");
263         sb.append(w.info.getName());
264         map.put("dcmd.help", executeDiagnosticCommand(sb.toString()));
265         if (w.info.getArgumentsInfo() != null && !w.info.getArgumentsInfo().isEm
266             HashMap<String, Object> allargmap = new HashMap<>();
267             for (DiagnosticCommandArgumentInfo arginfo : w.info.getArgumentsInfo
268                 HashMap<String, Object> argmap = new HashMap<>();
269                 argmap.put("dcmd.arg.name", arginfo.getName());
270                 argmap.put("dcmd.arg.type", arginfo.getType());
271                 argmap.put("dcmd.arg.description", arginfo.getDescription());
272                 argmap.put("dcmd.arg.isMandatory", arginfo.isMandatory());
273                 argmap.put("dcmd.arg.isMultiple", arginfo.isMultiple());
274                 boolean isOption = arginfo.isOption();
275                 argmap.put("dcmd.arg.isOption", isOption);
276                 if(!isOption) {
277                     argmap.put("dcmd.arg.position", arginfo.getPosition());
278                 } else {
279                     argmap.put("dcmd.arg.position", -1);
280                 }
281                 allargmap.put(arginfo.getName(), new ImmutableDescriptor(argmap)
282             }
283             map.put("dcmd.arguments", new ImmutableDescriptor(allargmap));
284         }
285         return new ImmutableDescriptor(map);
286     }

288     private final static String notifName =
289         "javax.management.Notification";

291     private final static String[] diagFramNotifTypes = {
292         "jmx.mbean.info.changed"
293     };

295     private MBeanNotificationInfo[] notifInfo = null;

297     @Override
298     public MBeanNotificationInfo[] getNotificationInfo() {
299         synchronized (this) {
300             if (notifInfo == null) {
301                  notifInfo = new MBeanNotificationInfo[1];
302                  notifInfo[0] =
303                         new MBeanNotificationInfo(diagFramNotifTypes,
304                                                   notifName,
305                                                   "Diagnostic Framework Notific
306             }
307         }
308         return notifInfo;
309     }

311     private static long seqNumber = 0;
312     private static long getNextSeqNumber() {
313         return ++seqNumber;
314     }

316     private void createDiagnosticFrameworkNotification() {

318         if (!hasListeners()) {
319             return;
320         }
321         ObjectName on = null;
322         try {
323             on = ObjectName.getInstance(ManagementFactoryHelper.HOTSPOT_DIAGNOST
324         } catch (MalformedObjectNameException e) { }
325         Notification notif = new Notification("jmx.mbean.info.changed",
326                                               on,
```



```
327                                               getNextSeqNumber());
328         notif.setUserData(getMBeanInfo());
329         sendNotification(notif);
330     }

332     @Override
333     public synchronized void addNotificationListener(NotificationListener listen
334             NotificationFilter filter,
335             Object handback) {
336         boolean before = hasListeners();
337         super.addNotificationListener(listener, filter, handback);
338         boolean after = hasListeners();
339         if (!before && after) {
340             setNotificationEnabled(true);
341         }
342     }

344     @Override
345     public synchronized void removeNotificationListener(NotificationListener lis
346             throws ListenerNotFoundException {
347         boolean before = hasListeners();
348         super.removeNotificationListener(listener);
349         boolean after = hasListeners();
350         if (before && !after) {
351             setNotificationEnabled(false);
352         }
353     }

355     @Override
356     public synchronized void removeNotificationListener(NotificationListener lis
357             NotificationFilter filter,
358             Object handback)
359             throws ListenerNotFoundException {
360         boolean before = hasListeners();
361         super.removeNotificationListener(listener, filter, handback);
362         boolean after = hasListeners();
363         if (before && !after) {
364             setNotificationEnabled(false);
365         }
366     }

368     private native void setNotificationEnabled(boolean enabled);
369     private native String[] getDiagnosticCommands();
370     private native DiagnosticCommandInfo[] getDiagnosticCommandInfo(String[] com
371     private native String executeDiagnosticCommand(String command);

373 }
```

```
**********************************************************
    5194 Fri May  3 09:28:12 2013
new/src/share/classes/sun/management/DiagnosticCommandInfo.java
**********************************************************
```

```
/*
 * Copyright (c) 2013, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Oracle designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Oracle in the LICENSE file that accompanied this code.
 *
 * This code is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 * or visit www.oracle.com if you need additional information or have any
 * questions.
 */

package sun.management;

import java.util.List;

/**
 * Diagnostic command information. It contains the description of a
 * diagnostic command.
 *
 * @since 8
 */

class DiagnosticCommandInfo {
    private final String name;
    private final String description;
    private final String impact;
    private final String permissionClass;
    private final String permissionName;
    private final String permissionAction;
    private final boolean enabled;
    private final List<DiagnosticCommandArgumentInfo> arguments;

    /**
     * Returns the diagnostic command name.
     *
     * @return the diagnostic command name
     */
    String getName() {
        return name;
    }

   /**
     * Returns the diagnostic command description.
     *
     * @return the diagnostic command description
     */
    String getDescription() {
        return description;
    }
```



```
    }

    /**
     * Returns the potential impact of the diagnostic command execution
     *         on the Java virtual machine behavior.
     *
     * @return the potential impact of the diagnostic command execution
     *         on the Java virtual machine behavior
     */
    String getImpact() {
        return impact;
    }

    /**
     * Returns the name of the permission class required to be allowed
     *         to invoke the diagnostic command, or null if no permission
     *         is required.
     *
     * @return the name of the permission class name required to be allowed
     *         to invoke the diagnostic command, or null if no permission
     *         is required
     */
    String getPermissionClass() {
        return permissionClass;
    }

    /**
     * Returns the permission name required to be allowed to invoke the
     *         diagnostic command, or null if no permission is required.
     *
     * @return the permission name required to be allowed to invoke the
     *         diagnostic command, or null if no permission is required
     */
    String getPermissionName() {
        return permissionName;
    }

    /**
     * Returns the permission action required to be allowed to invoke the
     *         diagnostic command, or null if no permission is required or
     *         if the permission has no action specified.
     *
     * @return the permission action required to be allowed to invoke the
     *         diagnostic command, or null if no permission is required or
     *         if the permission has no action specified
     */
    String getPermissionAction() {
        return permissionAction;
    }

    /**
     * Returns {@code true} if the diagnostic command is enabled,
     *         {@code false} otherwise. The enabled/disabled
     *         status of a diagnostic command can evolve during
     *         the lifetime of the Java virtual machine.
     *
     * @return {@code true} if the diagnostic command is enabled,
     *         {@code false} otherwise
     */
    boolean isEnabled() {
        return enabled;
    }

    /**
     * Returns the list of the diagnostic command arguments description.
     * If the diagnostic command has no arguments, it returns an empty list.
```

```
129      *
130      * @return a list of the diagnostic command arguments description
131      */
132     List<DiagnosticCommandArgumentInfo> getArgumentsInfo() {
133         return arguments;
134     }

136     DiagnosticCommandInfo(String name, String description,
137                                   String impact, String permissionClass,
138                                   String permissionName, String permissionActi
139                                   boolean enabled,
140                                   List<DiagnosticCommandArgumentInfo> argument
141     {
142         this.name = name;
143         this.description = description;
144         this.impact = impact;
145         this.permissionClass = permissionClass;
146         this.permissionName = permissionName;
147         this.permissionAction = permissionAction;
148         this.enabled = enabled;
149         this.arguments = arguments;
150     }
151 }
```

```
**********************************************************
    7201 Fri May  3 09:28:13 2013
new/src/share/native/sun/management/DiagnosticCommandImpl.c
**********************************************************
```

```
/*
 * Copyright (c)  2013, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Oracle designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Oracle in the LICENSE file that accompanied this code.
 *
 * This code is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 * or visit www.oracle.com if you need additional information or have any
 * questions.
 */

#include <jni.h>
#include "management.h"
#include "sun_management_DiagnosticCommandImpl.h"

JNIEXPORT void JNICALL Java_sun_management_DiagnosticCommandImpl_setNotification
(JNIEnv *env, jobject dummy, jboolean enabled) {
    if(jmm_version > JMM_VERSION_1_2_2) {
        jmm_interface->SetDiagnosticFrameworkNotificationEnabled(env, enabled);
    } else {
        JNU_ThrowByName(env, "java/lang/UnsupportedOperationException",
                        "JMX interface to diagnostic framework notifications is
    }
}

JNIEXPORT jobjectArray JNICALL
Java_sun_management_DiagnosticCommandImpl_getDiagnosticCommands
  (JNIEnv *env, jobject dummy)
{
  return jmm_interface->GetDiagnosticCommands(env);
}

jobject getDiagnosticCommandArgumentInfoArray(JNIEnv *env, jstring command,
                                              int num_arg) {
  int i;
  jobject obj;
  jobjectArray result;
  dcmdArgInfo* dcmd_arg_info_array;
  jclass dcmdArgInfoCls;
  jclass arraysCls;
  jmethodID mid;
  jobject resultList;

  dcmd_arg_info_array = (dcmdArgInfo*) malloc(num_arg * sizeof(dcmdArgInfo));
  if (dcmd_arg_info_array == NULL) {
    return NULL;
  }
  jmm_interface->GetDiagnosticCommandArgumentsInfo(env, command,
```



```
                                                   dcmd_arg_info_array);
  dcmdArgInfoCls = (*env)->FindClass(env,
                                     "sun/management/DiagnosticCommandArgumentIn
  result = (*env)->NewObjectArray(env, num_arg, dcmdArgInfoCls, NULL);
  if (result == NULL) {
    free(dcmd_arg_info_array);
    return NULL;
  }
  for (i=0; i<num_arg; i++) {
    obj = JNU_NewObjectByName(env,
                              "sun/management/DiagnosticCommandArgumentInfo",
                              "(Ljava/lang/String;Ljava/lang/String;Ljava/lang/S
                              (*env)->NewStringUTF(env,dcmd_arg_info_array[i].na
                              (*env)->NewStringUTF(env,dcmd_arg_info_array[i].de
                              (*env)->NewStringUTF(env,dcmd_arg_info_array[i].ty
                              dcmd_arg_info_array[i].default_string == NULL ? NU
                              (*env)->NewStringUTF(env, dcmd_arg_info_array[i].d
                              dcmd_arg_info_array[i].mandatory,
                              dcmd_arg_info_array[i].option,
                              dcmd_arg_info_array[i].multiple,
                              dcmd_arg_info_array[i].position);
    if (obj == NULL) {
      free(dcmd_arg_info_array);
      return NULL;
    }
    (*env)->SetObjectArrayElement(env, result, i, obj);
  }
  free(dcmd_arg_info_array);
  arraysCls = (*env)->FindClass(env, "java/util/Arrays");
  mid = (*env)->GetStaticMethodID(env, arraysCls,
                                  "asList", "([Ljava/lang/Object;)Ljava/util/Lis
  resultList = (*env)->CallStaticObjectMethod(env, arraysCls, mid, result);
  return resultList;
}

/* Throws IllegalArgumentException if at least one of the diagnostic command
 * passed in argument is not supported by the JVM
 */
JNIEXPORT jobjectArray JNICALL
Java_sun_management_DiagnosticCommandImpl_getDiagnosticCommandInfo
(JNIEnv *env, jobject dummy, jobjectArray commands)
{
  int i;
  jclass dcmdInfoCls;
  jobject result;
  jobjectArray args;
  jobject obj;
  jmmOptionalSupport mos;
  jint ret = jmm_interface->GetOptionalSupport(env, &mos);
  jsize num_commands;
  dcmdInfo* dcmd_info_array;

  if (commands == NULL) {
      JNU_ThrowNullPointerException(env, "Invalid String Array");
      return NULL;
  }
  num_commands = (*env)->GetArrayLength(env, commands);
  dcmd_info_array = (dcmdInfo*) malloc(num_commands *
                                       sizeof(dcmdInfo));
  if (dcmd_info_array == NULL) {
      JNU_ThrowOutOfMemoryError(env, NULL);
  }
  jmm_interface->GetDiagnosticCommandInfo(env, commands, dcmd_info_array);
  dcmdInfoCls = (*env)->FindClass(env,
                                  "sun/management/DiagnosticCommandInfo");
  result = (*env)->NewObjectArray(env, num_commands, dcmdInfoCls, NULL);
```

```
129   if (result == NULL) {
130       free(dcmd_info_array);
131       JNU_ThrowOutOfMemoryError(env, 0);
132   }
133   for (i=0; i<num_commands; i++) {
134       args = getDiagnosticCommandArgumentInfoArray(env,
135                                                    (*env)->GetObjectArrayElement
136                                                    dcmd_info_array[i].num_argume
137       if (args == NULL) {
138           free(dcmd_info_array);
139           JNU_ThrowOutOfMemoryError(env, 0);
140       }
141       obj = JNU_NewObjectByName(env,
142                                 "sun/management/DiagnosticCommandInfo",
143                                 "(Ljava/lang/String;Ljava/lang/String;Ljava/lang
144                                 (*env)->NewStringUTF(env,dcmd_info_array[i].name
145                                 (*env)->NewStringUTF(env,dcmd_info_array[i].desc
146                                 (*env)->NewStringUTF(env,dcmd_info_array[i].impa
147                                 dcmd_info_array[i].permission_class==NULL?NULL:(
148                                 dcmd_info_array[i].permission_name==NULL?NULL:(*
149                                 dcmd_info_array[i].permission_action==NULL?NULL:
150                                 dcmd_info_array[i].enabled,
151                                 args);
152       if (obj == NULL) {
153           free(dcmd_info_array);
154           JNU_ThrowOutOfMemoryError(env, 0);
155       }
156       (*env)->SetObjectArrayElement(env, result, i, obj);
157   }
158   free(dcmd_info_array);
159   return result;
160 }

162 /* Throws IllegalArgumentException if the diagnostic command
163  * passed in argument is not supported by the JVM
164  */
165 JNIEXPORT jstring JNICALL
166 Java_sun_management_DiagnosticCommandImpl_executeDiagnosticCommand
167 (JNIEnv *env, jobject dummy, jstring command) {
168   return jmm_interface->ExecuteDiagnosticCommand(env, command);
169 }
```

```
**********************************************************
    3499 Fri May  3 09:28:14 2013
new/test/com/sun/management/DiagnosticCommandMBean/DcmdMBeanDoubleInvocationTest
.java
**********************************************************
```

```
/*
 * Copyright (c) 2013, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.
 *
 * This code is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 * or visit www.oracle.com if you need additional information or have any
 * questions.
 */

/*
 * @test
 * @bug     7150256
 * @summary Basic Test for the DiagnosticCommandMBean
 * @author  Frederic Parain
 *
 * @run main/othervm -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun
 */


import java.io.IOException;
import java.lang.management.ManagementFactory;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.management.Descriptor;
import javax.management.InstanceNotFoundException;
import javax.management.IntrospectionException;
import javax.management.MBeanInfo;
import javax.management.MBeanOperationInfo;
import javax.management.MBeanServer;
import javax.management.MalformedObjectNameException;
import javax.management.ObjectName;
import javax.management.ReflectionException;
import javax.management.*;
import javax.management.remote.*;

public class DcmdMBeanDoubleInvocationTest {

    private static String HOTSPOT_DIAGNOSTIC_MXBEAN_NAME =
        "com.sun.management:type=DiagnosticCommand";

    public static void main(String[] args) {
        MBeanServerConnection mbs = null;
        try {
            JMXServiceURL url = new JMXServiceURL("service:jmx:rmi:///jndi/rmi:/
            JMXConnector connector = JMXConnectorFactory.connect(url);
            mbs = connector.getMBeanServerConnection();
        } catch(Throwable t) {
```



```
            t.printStackTrace();
        }
        ObjectName name;
        try {
            name = new ObjectName(HOTSPOT_DIAGNOSTIC_MXBEAN_NAME);
            MBeanInfo info = mbs.getMBeanInfo(name);
            String[] helpArgs = {"-all", "\n", "VM.version"};
            Object[] dcmdArgs = {helpArgs};
            String[] signature = {String[].class.getName()};
            String result = (String) mbs.invoke(name, "help", dcmdArgs, signatur
            System.out.println(result);
         } catch (RuntimeMBeanException ex) {
            if (ex.getCause() instanceof IllegalArgumentException) {
                System.out.println("Test passed");
                return;
            } else {
                ex.printStackTrace();
                throw new RuntimeException("TEST FAILED");
            }
        } catch (InstanceNotFoundException | IntrospectionException
                | ReflectionException | MalformedObjectNameException
                | MBeanException|IOException ex) {
            ex.printStackTrace();
            throw new RuntimeException("TEST FAILED");
        }
        System.out.println("Double commands have not been detected");
        throw new RuntimeException("TEST FAILED");
    }
}
```

```
**********************************************************
    3179 Fri May  3 09:28:15 2013
new/test/com/sun/management/DiagnosticCommandMBean/DcmdMBeanInvocationTest.java
**********************************************************
```

```
 1 /*
 2  * Copyright (c) 2013, Oracle and/or its affiliates. All rights reserved.
 3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 4  *
 5  * This code is free software; you can redistribute it and/or modify it
 6  * under the terms of the GNU General Public License version 2 only, as
 7  * published by the Free Software Foundation.
 8  *
 9  * This code is distributed in the hope that it will be useful, but WITHOUT
10  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
11  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
12  * version 2 for more details (a copy is included in the LICENSE file that
13  * accompanied this code).
14  *
15  * You should have received a copy of the GNU General Public License version
16  * 2 along with this work; if not, write to the Free Software Foundation,
17  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
18  *
19  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
20  * or visit www.oracle.com if you need additional information or have any
21  * questions.
22  */

24 /*
25  * @test
26  * @bug     7150256
27  * @summary Basic Test for the DiagnosticCommandMBean
28  * @author  Frederic Parain
29  *
30  * @run main/othervm -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun
31  */


34 import java.io.IOException;
35 import java.lang.management.ManagementFactory;
36 import java.util.logging.Level;
37 import java.util.logging.Logger;
38 import javax.management.Descriptor;
39 import javax.management.InstanceNotFoundException;
40 import javax.management.IntrospectionException;
41 import javax.management.MBeanInfo;
42 import javax.management.MBeanOperationInfo;
43 import javax.management.MBeanServer;
44 import javax.management.MalformedObjectNameException;
45 import javax.management.ObjectName;
46 import javax.management.ReflectionException;
47 import javax.management.*;
48 import javax.management.remote.*;

50 public class DcmdMBeanInvocationTest {

52     private static String HOTSPOT_DIAGNOSTIC_MXBEAN_NAME =
53         "com.sun.management:type=DiagnosticCommand";

55     public static void main(String[] args) {
56         MBeanServerConnection mbs = null;
57         try {
58             JMXServiceURL url = new JMXServiceURL("service:jmx:rmi:///jndi/rmi:/
59             JMXConnector connector = JMXConnectorFactory.connect(url);
60             mbs = connector.getMBeanServerConnection();
61         } catch(Throwable t) {
62             t.printStackTrace();
```



```
63         }
64         ObjectName name;
65         try {
66             name = new ObjectName(HOTSPOT_DIAGNOSTIC_MXBEAN_NAME);
67             MBeanInfo info = mbs.getMBeanInfo(name);
68             String[] helpArgs = {"-all"};
69             Object[] dcmdArgs = {helpArgs};
70             String[] signature = {String[].class.getName()};
71             String result = (String) mbs.invoke(name, "help", dcmdArgs, signatur
72             System.out.println(result);
73         } catch (InstanceNotFoundException | IntrospectionException
74                 | ReflectionException | MalformedObjectNameException
75                 | MBeanException|IOException ex) {
76             ex.printStackTrace();
77             throw new RuntimeException("TEST FAILED");
78         }
79         System.out.println("Test passed");
80     }
81 }
```

```
**********************************************************
    9927 Fri May  3 09:28:16 2013
new/test/com/sun/management/DiagnosticCommandMBean/DcmdMBeanPermissionsTest.java
**********************************************************
```

```
 1 /*
 2  * Copyright (c) 2013, Oracle and/or its affiliates. All rights reserved.
 3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 4  *
 5  * This code is free software; you can redistribute it and/or modify it
 6  * under the terms of the GNU General Public License version 2 only, as
 7  * published by the Free Software Foundation.
 8  *
 9  * This code is distributed in the hope that it will be useful, but WITHOUT
10  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
11  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
12  * version 2 for more details (a copy is included in the LICENSE file that
13  * accompanied this code).
14  *
15  * You should have received a copy of the GNU General Public License version
16  * 2 along with this work; if not, write to the Free Software Foundation,
17  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
18  *
19  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
20  * or visit www.oracle.com if you need additional information or have any
21  * questions.
22  */

24 /*
25  * @test
26  * @bug     7150256
27  * @summary Permissions Tests for the DiagnosticCommandMBean
28  * @author  Frederic Parain
29  *
30  * @run main/othervm DcmdMBeanPermissionsTest
31  */

33 import java.lang.management.ManagementFactory;
34 import java.lang.reflect.Constructor;
35 import java.lang.reflect.InvocationTargetException;
36 import java.lang.reflect.ReflectPermission;
37 import java.security.Permission;
38 import java.util.HashSet;
39 import java.util.Iterator;
40 import javax.management.Descriptor;
41 import javax.management.InstanceNotFoundException;
42 import javax.management.IntrospectionException;
43 import javax.management.MBeanException;
44 import javax.management.MBeanInfo;
45 import javax.management.MBeanOperationInfo;
46 import javax.management.MBeanPermission;
47 import javax.management.MBeanServer;
48 import javax.management.MalformedObjectNameException;
49 import javax.management.ObjectName;
50 import javax.management.ReflectionException;
51 import javax.management.RuntimeMBeanException;

53 /**
54  *
55  * @author fparain
56  */
57 public class DcmdMBeanPermissionsTest {

59
60     private static String HOTSPOT_DIAGNOSTIC_MXBEAN_NAME =
61         "com.sun.management:type=DiagnosticCommand";
62
```



```
63     static public class CustomSecurityManager extends SecurityManager {

65         private HashSet<Permission> grantedPermissions;

67         public CustomSecurityManager() {
68             grantedPermissions = new HashSet<Permission>();
69         }

71         public final void grantPermission(final Permission perm) {
72             grantedPermissions.add(perm);
73         }

75         public final void denyPermission(final Permission perm) {
76             Iterator<Permission> it = grantedPermissions.iterator();
77             while (it.hasNext()) {
78                 Permission p = it.next();
79                 if (p.equals(perm)) {
80                     it.remove();
81                 }
82             }
83         }

85         public final void checkPermission(final Permission perm) {
86             for (Permission p : grantedPermissions) {
87                 if (p.implies(perm)) {
88                     return;
89                 }
90             }
91             throw new SecurityException(perm.toString());
92         }
93     };

95     static Permission createPermission(String classname, String name,
96             String action) {
97         Permission permission = null;
98         try {
99             Class c = Class.forName(classname);
100            if (action == null) {
101                try {
102                    Constructor constructor = c.getConstructor(String.class);
103                    permission = (Permission) constructor.newInstance(name);

105                } catch (InstantiationException | IllegalAccessException
106                        | IllegalArgumentException | InvocationTargetException
107                        | NoSuchMethodException | SecurityException ex) {
108                    ex.printStackTrace();
109                    throw new RuntimeException("TEST FAILED");
110                }
111            }
112            if (permission == null) {
113                try {
114                    Constructor constructor = c.getConstructor(String.class,
115                            String.class);
116                    permission = (Permission) constructor.newInstance(
117                            name,
118                            action);
119                } catch (InstantiationException | IllegalAccessException
120                        | IllegalArgumentException | InvocationTargetException
121                        | NoSuchMethodException | SecurityException ex) {
122                    ex.printStackTrace();
123                    throw new RuntimeException("TEST FAILED");
124                }
125            }
126        } catch (ClassNotFoundException ex) {
127            ex.printStackTrace();
128                    throw new RuntimeException("TEST FAILED");
```

```
129         }
130         if (permission == null) {
131             throw new RuntimeException("TEST FAILED");
132 
133         }
134         return permission;
135     }
136 
137     // return true if invokation triggered a SecurityException
138     static boolean invokeOperation(MBeanServer mbs, ObjectName on,
139             MBeanOperationInfo opInfo) {
140         try {
141             if (opInfo.getSignature().length == 0) {
142                 mbs.invoke(on, opInfo.getName(),
143                         new Object[0], new String[0]);
144             } else {
145                 mbs.invoke(on, opInfo.getName(),
146                         new Object[1], new String[]{ String[].class.getName()});
147             }
148         } catch (SecurityException ex) {
149             ex.printStackTrace();
150             return true;
151         } catch (RuntimeMBeanException ex) {
152             if (ex.getCause() instanceof SecurityException) {
153                 //ex.printStackTrace();
154                 return true;
155             }
156         } catch (MBeanException | InstanceNotFoundException
157                 | ReflectionException ex) {
158             throw new RuntimeException("TEST FAILED");
159         }
160         return false;
161     }
162 
163     static void testOperation(MBeanServer mbs, CustomSecurityManager sm,
164             ObjectName on, MBeanOperationInfo opInfo) {
165         System.out.println("Testing " + opInfo.getName());
166         Descriptor desc = opInfo.getDescriptor();
167         if (desc.getFieldValue("dcmd.permissionClass") == null) {
168         // No special permission required, execution should not trigger
169         // any security exception
170             if (invokeOperation(mbs, on, opInfo)) {
171                 throw new RuntimeException("TEST FAILED");
172             }
173         } else {
174             // Building the required permission
175             Permission reqPerm = createPermission(
176                     (String)desc.getFieldValue("dcmd.permissionClass"),
177                     (String)desc.getFieldValue("dcmd.permissionName"),
178                     (String)desc.getFieldValue("dcmd.permissionAction"));
179             // Paranoid mode: check that the SecurityManager has not already
180             // been granted the permission
181             sm.denyPermission(reqPerm);
182             // A special permission is required for this operation,
183             // invoking it without the permission granted must trigger
184             // a security exception
185             if(!invokeOperation(mbs, on, opInfo)) {
186                 throw new RuntimeException("TEST FAILED");
187             }
188             // grant the permission and re-try invoking the operation
189             sm.grantPermission(reqPerm);
190             if(invokeOperation(mbs, on, opInfo)) {
191                 throw new RuntimeException("TEST FAILED");
192             }
193             // Clean up
194             sm.denyPermission(reqPerm);
```



```
195         }
196     }
197 
198     public static void main(final String[] args) {
199         final MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();
200         ObjectName on = null;
201         try {
202             on = new ObjectName(HOTSPOT_DIAGNOSTIC_MXBEAN_NAME);
203         } catch (MalformedObjectNameException ex) {
204             ex.printStackTrace();
205             throw new RuntimeException("TEST FAILED");
206         }
207         MBeanInfo info = null;
208         try {
209             info = mbs.getMBeanInfo(on);
210         } catch (InstanceNotFoundException | IntrospectionException
211                 | ReflectionException ex) {
212             ex.printStackTrace();
213             throw new RuntimeException("TEST FAILED");
214         }
215         CustomSecurityManager sm = new CustomSecurityManager();
216         System.setSecurityManager(sm);
217         // Set of permission required to run the test cleanly
218         // Some permissions are required by the MBeanServer and other
219         // platform services (RuntimePermission("createClassLoader"),
220         // ReflectPermission("suppressAccessChecks"),
221         // java.util.logging.LoggingPermission("control"),
222         // RuntimePermission("exitVM.97")).
223         // Other permissions are required by commands being invoked
224         // in the test (for instance, RuntimePermission("modifyThreadGroup")
225         // and RuntimePermission("modifyThread") are checked when
226         // runFinalization() is invoked by the gcRunFinalization command.
227         sm.grantPermission(new RuntimePermission("createClassLoader"));
228         sm.grantPermission(new ReflectPermission("suppressAccessChecks"));
229         sm.grantPermission(new java.util.logging.LoggingPermission("control", ""
230         sm.grantPermission(new java.lang.RuntimePermission("exitVM.97"));
231         sm.grantPermission(new java.lang.RuntimePermission("modifyThreadGroup"))
232         sm.grantPermission(new java.lang.RuntimePermission("modifyThread"));
233         for(MBeanOperationInfo opInfo : info.getOperations()) {
234             Permission opPermission = new MBeanPermission(info.getClassName(),
235                     opInfo.getName(),
236                     on,
237                     "invoke");
238             sm.grantPermission(opPermission);
239             testOperation(mbs, sm, on, opInfo);
240             sm.denyPermission(opPermission);
241         }
242         System.out.println("TEST PASSED");
243     }
244 }
```

```
**********************************************************
    4834 Fri May  3 09:28:17 2013
new/test/com/sun/management/DiagnosticCommandMBean/DcmdMBeanTest.java
**********************************************************
```

```
 1 /*
 2  * Copyright (c) 2013, Oracle and/or its affiliates. All rights reserved.
 3  * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 4  *
 5  * This code is free software; you can redistribute it and/or modify it
 6  * under the terms of the GNU General Public License version 2 only, as
 7  * published by the Free Software Foundation.
 8  *
 9  * This code is distributed in the hope that it will be useful, but WITHOUT
10  * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
11  * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
12  * version 2 for more details (a copy is included in the LICENSE file that
13  * accompanied this code).
14  *
15  * You should have received a copy of the GNU General Public License version
16  * 2 along with this work; if not, write to the Free Software Foundation,
17  * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
18  *
19  * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
20  * or visit www.oracle.com if you need additional information or have any
21  * questions.
22  */

24 /*
25  * @test
26  * @bug     7150256
27  * @summary Basic Test for the DiagnosticCommandMBean
28  * @author  Frederic Parain
29  *
30  * @run main/othervm -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun
31  */


34 import java.io.IOException;
35 import java.lang.management.ManagementFactory;
36 import java.util.logging.Level;
37 import java.util.logging.Logger;
38 import javax.management.Descriptor;
39 import javax.management.InstanceNotFoundException;
40 import javax.management.IntrospectionException;
41 import javax.management.MBeanInfo;
42 import javax.management.MBeanOperationInfo;
43 import javax.management.MBeanServer;
44 import javax.management.MalformedObjectNameException;
45 import javax.management.ObjectName;
46 import javax.management.ReflectionException;
47 import javax.management.*;
48 import javax.management.remote.*;

50 public class DcmdMBeanTest {

52     private static String HOTSPOT_DIAGNOSTIC_MXBEAN_NAME =
53         "com.sun.management:type=DiagnosticCommand";

55     public static void main(String[] args) {
56         MBeanServerConnection mbs = null;
57         try {
58             JMXServiceURL url = new JMXServiceURL("service:jmx:rmi:///jndi/rmi:/
59             JMXConnector connector = JMXConnectorFactory.connect(url);
60             mbs = connector.getMBeanServerConnection();
61         } catch(Throwable t) {
62             t.printStackTrace();
```



```
63         }
64         ObjectName name;
65         try {
66             name = new ObjectName(HOTSPOT_DIAGNOSTIC_MXBEAN_NAME);
67             MBeanInfo info = mbs.getMBeanInfo(name);
68             // the test should check that the MBean doesn't have any
69             // Attribute, notification or constructor. Current version only
70             // check operations
71             System.out.println("Class Name:"+info.getClassName());
72             System.out.println("Description:"+info.getDescription());
73             MBeanOperationInfo[] opInfo = info.getOperations();
74             System.out.println("Operations:");
75             for(int i=0; i<opInfo.length; i++) {
76                 printOperation(opInfo[i]);
77                 System.out.println("\n@@@@@@\n");
78             }
79         } catch (InstanceNotFoundException|IntrospectionException|ReflectionExce
80                  |MalformedObjectNameException|IOException ex) {
81             Logger.getLogger(DcmdMBeanTest.class.getName()).log(Level.SEVERE, nu
82         }
83     }
84
85     static void printOperation(MBeanOperationInfo info) {
86         System.out.println("Name: "+info.getName());
87         System.out.println("Description: "+info.getDescription());
88         System.out.println("Return Type: "+info.getReturnType());
89         System.out.println("Impact: "+info.getImpact());
90         // print parameters info?
91         Descriptor desc = info.getDescriptor();
92         System.out.println("Descriptor");
93         for(int i=0; i<desc.getFieldNames().length; i++) {
94             if(desc.getFieldNames()[i].compareTo("dcmd.arguments") == 0) {
95                 System.out.println("\t"+desc.getFieldNames()[i]+":");
96                 Descriptor desc2 =
97                         (Descriptor)desc.getFieldValue(desc.getFieldNames()[i]);
98                 for(int j=0; j<desc2.getFieldNames().length; j++) {
99                     System.out.println("\t\t"+desc2.getFieldNames()[j]+"=");
100                    Descriptor desc3 =
101                            (Descriptor)desc2.getFieldValue(desc2.getFieldNames(
102                    for(int k=0; k<desc3.getFieldNames().length; k++) {
103                        System.out.println("\t\t\t"+desc3.getFieldNames()[k]+"="
104                                           +desc3.getFieldValue(desc3.getFieldNa
105                    }
106                }
107            } else {
108                System.out.println("\t"+desc.getFieldNames()[i]+"="
109                        +desc.getFieldValue(desc.getFieldNames()[i]));
110            }
111        }
112    }
113
114 }
```