Introduction

Oracle WebLogic Server is the #1 application server across conventional and cloud environments. As a JEE-compliant server with enterprise-class features, WebLogic Server is the premier option for mission-critical and highly-available applications. Leading Independent Software Vendors (ISVs) and System Integrators building their own IP use WebLogic Server along with Oracle Database as the core deployment platform for their Java Applications. Oracle WebLogic Server provides capabilities which allow ISVs to accelerate the development cycle and reduce time-to-market for their applications while also lowering the total cost of ownership for their customers. Oracle WebLogic Server is also the foundation for the Oracle Fusion Middleware technology stack - the only middleware available from any vendor that provides a complete, open and integrated approach across social, mobile, and cloud technologies.

ISVs license Oracle WebLogic Server under different distribution agreements but one of the most popular programs from Oracle is the Embedded Software License (ESL) Distribution Agreement. Embedding is the processes of packaging WebLogic Server within an ISV application in a manner that makes the presence of WebLogic Server invisible to the end user of the application. This allows ISVs to provide WebLogic Server as a “black box”, reducing the support overhead associated with end customers making unwarranted changes to the technology infrastructure. Further benefits include simplifying the installation & maintenance of the application and the underlying technology infrastructure. Oracle also has aggressive discounts available under the ESL program.

This white paper is intended to guide a partner through the Oracle WebLogic Server embedding process with an eye towards ESL compliance. Use this paper in combination with sample scripts in the Appendix and documentation pointers to create your embedding strategy. Documentation links are for Oracle Fusion Middleware 11gR1. Version specific documentation for Oracle Fusion Middleware is available on OTN http://www.oracle.com/technetwork/middleware. Similarly, scripts should be edited & tested based on specific deployment environment.

Embedding WebLogic Overview

The technical process of embedding involves encapsulating the installation and management of the embedded components through the use of silent installation & API/script based management techniques. Embedding WebLogic Server involves planning 4 different areas –

- Installing WebLogic Server silently with ISV Application
- Creating & Configuring WebLogic Server Domains through scripts
- Deploying the ISV Application on WebLogic Server through scripts
- Managing & Monitoring WebLogic Domain & Servers through scripts or java code which can be called if needed from the ISV’s Application Management Console.
Patching of the embedded WebLogic Server should be done using the SmartUpdate tool. Other Oracle Fusion Middleware components use OPatch and in the future WebLogic Server will also support the OPatch utility.

Silent Installation

Silent installation of WebLogic Server is typically embedding requirement number one. An overview of WebLogic installation is available in the installation guide. Silent installations have been part of the installation and deployment strategy at Oracle for many years. Simply stated, silent installation is a method by which Oracle WebLogic server is installed in a manner that requires zero interaction by the person doing the installation. In a silent installation, no installation forms or feedback screen is given. All options and configurations are controlled in xml file(s). In a situation involving an embedded software license, the partner application should call the silent installation directly from within its own installer, thus completely hiding the WebLogic Server installation. A silent installation is not unlike a standard installation as it is dependent on system prerequisites.

The silent.xml file is the main driver of the silent installation process. It is where one configures how WebLogic Server is installed. This file can be parameterized to support inputs from another installer such as an ISV’s installation process thus allowing input from those forms to dynamically fill out a silent.xml for use in silent installation. A sample silent.xml is available here.

Without replicating the comprehensive documentation linked earlier, suffice it to say there are three ways to start a silent installation.

- A bin installation for *NIX platforms
- A jar installation for generic platform installations (perhaps the easiest if your application runs on many different platforms)
- An exe installation built for Windows targets

Each installation is the same package one would use for an interactive installation with the addition of a flag identifying it as silent (-mode=silent) and a pointer to your silent.xml (-silent_xml=<path>). The following link describes the installer files used for each platform. Please remember each could be used for interactive as well as silent installations so flagging the mode is critical.

Domain Creation & Configuration

The WebLogic Scripting Tool (WLST) is a command-line scripting environment that you can use to create, manage, and monitor WebLogic Server domains. It is based on the Java scripting interpreter, Jython. You can also instantiate WLST in Java code and use it to run WLST commands and scripts. WLST enables you to create a new WebLogic domain or update an existing WebLogic domain without connecting to a running WebLogic Server. This can be done in two ways -
• **Creating and Using a Domain Template (Offline)**

A configuration template defines the full set of resources within a domain, including infrastructure components, applications, services, security options, and general environment and operating system options. Oracle provides a number of templates and template extensions as part of the WebLogic Platform product. Appendix A provides a WLST offline configuration script to create a simple clustered WebLogic domain using the Basic WebLogic Server Domain template. Domain Template Builder is a standalone Java tool to create templates which can then be applied in the offline creation of WebLogic Server domains. An ISV can create a standard domain template using Domain Template Builder. Domain Template Builder accomplishes template creation based on an interactive GUI, pack and unpack provide a scriptable, command-line utility.

• **Using the configToScript Command**

WLST includes a command, configToScript, which reads an existing WebLogic domain and creates a WLST script that can recreate the WebLogic domain. This option requires a running WebLogic Server instance.

**Application Deployment**

Application deployment on WebLogic Server is the process of distributing an application for server-side processing and application startup. Oracle Fusion Middleware deploying applications to WebLogic Server guide provides in depth documentation for all aspects of deployment.

WebLogic server can support various deployment units: enterprise application, web application, EJB, resource adapter, web service, Java EE library, JDBC, JMS, WLDF modules and so on. Before deploying, we need to prepare applications and modules for deployment. WebLogic Server supports deployments that are packaged either as archive files (.ear, .jar, .war, and so forth) using the jar utility, or as exploded archive directories. In stable production environment, we would recommend to use archive files, since they are more efficient for deploying applications to managed servers. Although you can deploy to the Administration Server in a multiple-server domain, this practice is not recommended except during development.

The WebLogic Administration Console, weblogic.Deployer tool, wlddeploy Ant task, and WLST all provide similar functionality for deploying applications. Use WLST when you want to create automated scripts that perform deployment tasks. Use weblogic.Deployer to integrate deployment commands with existing administrative shell scripts or automated batch processes. Use wlddeploy in conjunction with the split development directory for developing and deploying new applications. wlddeploy can also be used in place of weblogic.Deployer in administration environments that use Ant, rather than shell script.

**Deploying Applications with WLST**

We can use either WLST online or offline to deploy applications:
• **WLST offline** - If the WebLogic Server is not running, for example the WebLogic domain was just created or the Admin server and managed servers are not started yet. In this case, we have to use WLST offline. A sample script is available in Appendix B.

• **WLST online** - If the WebLogic Server is running, this method is highly recommended. A sample script is available in Appendix C.

Deploying Applications with weblogic.Deployer

weblogic.Deployer provides a command-line based interface for performing both basic and advanced deployment tasks. Common Deployment Scenarios with weblogic.Deployer is available [here](#).

weblogic.Deployer commands and command options used to perform deployment tasks with WebLogic Server are available [here](#).

Example –

C:\>java weblogic.Deployer -adminurl t3://localhost:7001 -username weblogic -password welcome1 -name ShoppingCart -targets examplesServer -deploy C:\ShoppingCart.war

The **connection arguments** are “-adminurl t3://localhost:7001”; the user credential arguments are organized as “-username weblogic -password welcome1”. In this sample command, the command arguments used are “deploy”, “targets” and “name”.

Server Administration

Management

During its lifetime, a server can transition through a number of operational states, such as shutdown, starting, standby, admin, resuming, and running. Server lifecycle management in an embedded environment requires non-interactive, script-based management techniques. Silent stopping and starting of domain servers such as administration and managed servers is typically just a matter of script execution in a Linux and UNIX environment. The same is true of Windows environments as well but with the option of creating a service to start and stop the servers. The following [link](#) describes how to start and stop WebLogic Server using WLST. Similarly, other Server operations can be done using WLST -

• Appendix D illustrates how to instantiate the WLST interpreter in java and use it to connect to a running server, create two servers, and assign them to cluster.

• Appendix E provides steps to change server runtime parameters.

Monitoring

Java Management Extensions (JMX) is a specification for monitoring and managing Java applications. It enables a generic management system to monitor your application; raise notifications when the application needs attention; change the state of your application to remedy problems. A managed bean
(MBean) is a Java bean that provides a Java Management Extensions (JMX) interface. WebLogic (WL) provides MBeans for both admin and managed servers. Sample code to read server information is provided in Appendix F. WebLogic Server also provides RESTful Services to monitor Server, clusters, Applications, and Data Sources. More details are available here. Partners can use the JMX or REST APIs to implement customized monitoring solution for WebLogic.

Conclusion

Embedded WebLogic have many benefits, not only for ISVs and OEMs developing a business application, but also for their end users or customers. Embedded WebLogic enables ISVs to build critical technology into their products for higher cost savings, improved management control, and greater efficiency. For end users, embedded WebLogic offer high performance and rapid installation with lower IT and support costs.

This white paper outlined steps for embedding Oracle WebLogic Sever. A similar white paper for embedding Oracle Database is available here on OTN. While this document addresses Oracle WebLogic Server, other components of Oracle Fusion Middleware can be embedded in a similar manner. The benefit to ISV partners and their customers in the adoption the Oracle Fusion Middleware stack include fast time to market and lower total cost of ownership.
Appendix

Appendix A – Domain Creation & Configuration

# This is an example of a simple WLST offline configuration script. The script creates
# a simple clustered WebLogic domain using the Basic WebLogic Server Domain template. The script
# demonstrates how to open a domain template, create and edit configuration objects,
# and write the domain configuration information to the specified directory.
#
# This script is an adaptation of several delivered sample WLST scripts that come with
# your WLS installation as well as some code from additional sources.
#
# (WL_HOME refers to the top level installation directory for WebLogic Server.)
#
# The sample consists of three managed servers, representing a simple single-node cluster for a development environment.
# Use of this script without editing and strenuous testing is not recommended (and is NOT SUPPORTED) for production environments.
#
# Please note that some of the values used in this script are subject to change based on
# your WebLogic installation and the template you are using.
#
# Usage:
#   java weblogic.WLST <WLST_script>
#
# Where:
#   <WLST_script> specifies the full path to the WLST script.
#
# 22 March 2012
# Chuck Speaks - Oracle North American ISV/OEM Sales / Worldwide Alliance and Channels

# Open a domain template.
readTemplate("/u01/oracle/middleware/wlserver_10.3/common/templates/domains/wls.jar")

# Configure the Administration Server and SSL port.
# To enable access by both local and remote processes, you should not set the
# listen address for the server instance (that is, it should be left blank or not set).
# In this case, the server instance will determine the address of the machine and
# listen on it.
#=======================================================================================

cd('Servers/AdminServer')
set('ListenAddress','')
set('ListenPort', 7001)

create('AdminServer','SSL')
cd('SSL/AdminServer')
set('Enabled', 'True')
set('ListenPort', 7002)

# Define the user password for weblogic.
#=======================================================================================

cd('/')
cd('Security/base_domain/User/weblogic')
# Please set password here before using this script, e.g. cmo.setPassword('value')
cmo.setPassword('Welcome1')

# Set Options:
# - CreateStartMenu: Enable creation of Start Menu shortcut.
# - ServerStartMode: Set mode to development.
# - JavaHome: Sets home directory for the JVM used when starting the server.
# - OverwriteDomain: Overwrites domain, when saving, if one exists.
#=======================================================================================

setOption('CreateStartMenu', 'false')
setOption('ServerStartMode', 'dev')
setOption('JavaHome','/u01/oracle/middleware/jrockit_160_24_D1.1.2-4')
setOption('OverwriteDomain', 'true')

# Create a Machine.
cd('/')
create('embeddedMachine-0','UnixMachine')
cd('/Machines/embeddedMachine-0')
#create('embeddedMachine-0-NM','NodeManager')
#cd('/NodeManager/embeddedMachine-0-NM')
#set('ListenAddress','embeddedfmw.oracle.com')
#set('NMType','SSL')
#set('ListenPort','5556')
activate()

# Start the Node Manager.
startNodeManager()

# Create a JMS Server.
cd('/')
create('myJMSServer', 'JMSServer')

# Create a JMS System resource.
cd('/')
create('myJmsSystemResource', 'JMSSystemResource')
cd('JMSSystemResource/myJmsSystemResource/JmsResource/NO_NAME_0')

# Create a JMS Queue and its subdeployment.
```java
myq=create('myQueue','Queue')
myq.setJNDIName('jms/myqueue')
myq.setSubDeploymentName('myQueueSubDeployment')

cd('/')
cd('JMSSystemResource/myJmsSystemResource')
create('myQueueSubDeployment', 'SubDeployment')

#=======================================================================================
# Create and configure a JDBC Data Source, and sets the JDBC user.
#=======================================================================================

#cd('/')
#create('myDataSource', 'JDBCSystemResource')
#cd('JDBCSystemResource/myDataSource/JdbcResource/myDataSource')
#create('myJdbcDriverParams','JDBCDriverParams')
#cd('JDBCDriverParams/NO_NAME_0')
#set('DriverName','com.pointbase.jdbc.jdbcUniversalDriver')
#set('URL','jdbc:pointbase:server://localhost/demo')
#set('PasswordEncrypted', 'PBPUBLIC')
#set('UseXADataSourceInterface', 'false')
#create('myProps','Properties')
#cd('Properties/NO_NAME_0')
#create('user', 'Property')
#cd('Property/user')
#cmo.setValue('PBPUBLIC')

#cd('JDBCSystemResource/myDataSource/JdbcResource/myDataSource')
#create('myJdbcDataSourceParams','JDBCDataSourceParams')
#cd('JDBCDataSourceParams/NO_NAME_0')
#set('JNDIName', java.lang.String("myDataSource_jndi"))

#cd('JDBCSystemResource/myDataSource/JdbcResource/myDataSource')
#create('myJdbcConnectionPoolParams','JDBCColocationPoolParams')
#cd('JDBCColocationPoolParams/NO_NAME_0')
#set('TestTableName','SYSTABLES')

#=======================================================================================
# Target resources to the servers.
```
```bash
#cd('/')
#assign('JMSServer', 'myJMSServer', 'Target', 'AdminServer')
#assign('JMSSystemResource.SubDeployment', 'myJmsSystemResource.myQueueSubDeployment', 'Target', 'myJMSServer')
#assign('JDBCSystemResource', 'myDataSource', 'Target', 'AdminServer')

#cd('/')
create('embeddedServer1', 'Server')
cd('Server/embeddedServer1')
set('ListenPort', 8001)
set('ListenAddress', '')
set('Machine', 'embeddedMachine-0')

cd('/')
create('embeddedServer2', 'Server')
cd('Server/embeddedServer2')
set('ListenPort', 8011)
set('ListenAddress', '')
set('Machine', 'embeddedMachine-0')

cd('/')
create('embeddedServer3', 'Server')
cd('Server/embeddedServer3')
set('ListenPort', 8021)
set('ListenAddress', '')
set('Machine', 'embeddedMachine-0')

#cd('/')
create('embeddedServer1', 'Server')
cd('Server/embeddedServer1')
set('ListenPort', 8001)
set('ListenAddress', '')
set('Machine', 'embeddedMachine-0')

cd('/')
create('embeddedServer2', 'Server')
cd('Server/embeddedServer2')
set('ListenPort', 8011)
set('ListenAddress', '')
set('Machine', 'embeddedMachine-0')

cd('/')
create('embeddedServer3', 'Server')
cd('Server/embeddedServer3')
set('ListenPort', 8021)
set('ListenAddress', '')
set('Machine', 'embeddedMachine-0')
```

# Create three Managed Servers and configure them.

# Migratable servers, which provide for both automatic and manual migration
# at the server-level, are created automatically when you create the Managed Servers.

# Create and configure a cluster and assign the Managed Servers to that cluster.
# Embedding Oracle Fusion Middleware

```bash
#======================================================================================
cd('/')
create('embeddedCluster', 'Cluster')
assign('Server', 'embeddedServer1,embeddedServer2,embeddedServer3','Cluster','embeddedCluster')
cd('Cluster/embeddedCluster')
set('MulticastAddress', '237.0.0.101')
set('MulticastPort', 8050)
set('WeblogicPluginEnabled', 'true')

#=======================================================================================
# Enroll Machine with Node Manager.
#=======================================================================================
#connect('weblogic','Welcome1','t3://embeddedfmw.oracle.com:7001')
#nmEnroll('/u01/oracle/middleware/user_projects/domains/baseWLSDomain','/u01/oracle/middleware/wlservr_10.3/server/bin')

#=======================================================================================
# Write the domain and close the domain template.
#=======================================================================================
setOption('OverwriteDomain', 'true')
writeDomain('/u01/oracle/middleware/user_projects/domains/basicWLSDomain')
closeTemplate()

#=======================================================================================
# Exit WLST.
#=======================================================================================
exit()
```
Appendix B – Deploy Application Offline Mode

# This is an example of a simple WLST offline application deployment script.
# Create a new application deployment
#
# Usage: java weblogic.WLST deploy_app_offline.py
#
#=======================================================================================
# Read domain configurations, deploy application in offline mode.
#=======================================================================================

readDomain(r'C:\Oracle\Middleware\wlserver_10.3\samples\domains\wl_server')
cd('/')
myApp=create('ShoppingCart', 'AppDeployment')
myApp.setSourcePath('C:/ShoppingCart.war')
assign('AppDeployment', 'ShoppingCart', 'Target', 'examplesServer')

#=======================================================================================
# Write the domain and close the domain template.
#=======================================================================================

updateDomain()
closeDomain()

#=======================================================================================
# Exit WLST.
#=======================================================================================

exit()
# This is an example of a simple WLST oneline application deployment script.
# In this script, it will attempt to connect to a given administration server, and deploy an application.
# Usage: java weblogic.WLST deploy_app_online.py. The .py file is a plain text file, which can support inputs from other program, for example, ISV's installation cab.

```python
# Set the parameter
url = 't3://localhost:7001'
username = 'weblogic'
password = 'welcome1'
appname = 'ShoppingCart'
appsource = 'C:/ShoppingCart.war'
target='examplesServer'

# Connect to Weblogic Server
try:
    connect(username, password, url)
except WLSTException:
    print 'Can not connect to server.'
    exit()

# Deploy the application
progress = deploy(appName=appname,path=appsource,targets=target)

# Exit WLST
print 'Application ' + appname + ' deployed.'
exit()
```

Appendix C – Deploy Application Online Mode
Appendix D – Create Managed Server and Associate to Cluster

```java
package wlstest;

import java.util.*;
import weblogic.management.scripting.utils.WLSTInterpreter;
import org.python.util.InteractiveInterpreter;

public class WLSTCluster {

    static InteractiveInterpreter interpreter = null;

    public WLSTCluster() {
        super();
        interpreter = new WLSTInterpreter();
    }

    private static void connect() {
        StringBuffer buffer = new StringBuffer();
        System.out.printf("connect to WLS using WLST...
    ");
        buffer.append("connect('weblogic', 'zhang1977223YANG', 't3://127.0.0.1:7001')");
        interpreter.exec(buffer.toString());
        System.out.printf("connect success...
    ");
    }

    private static void createServers() {
        StringBuffer buf = new StringBuffer();
        buf.append(startTransaction());
        buf.append("man7=create('ManagedServer_7', 'Server')
    ");
        buf.append("man7.setListenPort(7007)
    ");
        buf.append("man7.setListenAddress('127.0.0.1')
    ");
        buf.append("cd('/')
    ");
        buf.append("cd('Servers/ManagedServer_7')
    ");
        buf.append("set('Machine', 'Machine_1')
    ");
        buf.append("cd('/')
    ");
        buf.append("cd('Clusters/Cluster_1')
    ");
        buf.append("man7.setCluster(cmo)
    ");
        //buf.append("assign('Server', 'ManagedServer_7', 'Cluster', 'My_Cluster_1')
    ");
```
buf.append("cd('/')\n");
buf.append("man8=create('ManagedServer_8','Server')\n");
buf.append("man8.setListenPort(7008)\n");
buf.append("man8.setListenAddress('127.0.0.1')\n");
buf.append("man8.setMachine('Machine_1')\n");
buf.append("cd('/')\n");
buf.append("cd('Clusters/Cluster_1')\n");
buf.append("man8.setCluster(cmo)\n");

//buf.append("clus2=create('Cluster-2','Cluster')\n");
//buf.append("man7.setCluster(clus2)\n");
//buf.append("man8.setCluster(clus2)\n");

buf.append(endTransaction());

buf.append("start('ManagedServer_7','Server','t3://127.0.0.1:7001')\n");
interpreter.exec(buf.toString());
}

private static String startTransaction() {
    StringBuffer buf = new StringBuffer();
    buf.append("edit()\n");
    buf.append("startEdit()\n");
    return buf.toString();
}

private static String endTransaction() {
    StringBuffer buf = new StringBuffer();
    buf.append("save()\n");
    buf.append("activate(block='true')\n");
    return buf.toString();
}

public static void main(String[] args) {
    WLSTCluster wLSTCluster = new WLSTCluster();
    connect();
    createServers();

    }
}
Appendix E – Change Server Runtime Parameters

```java
package wlstest;
import weblogic.management.scripting.utils.WLSTInterpreter;
import org.python.util.InteractiveInterpreter;
...

# connect to server
buffer.append("connect('weblogic', 'zhang1977223YANG', 't3://127.0.0.1:7001')");
interpreter.exec(buffer.toString());

# Change runtime parameters
buf.append("edit()
");
buf.append("startEdit()
");
buf.append("cd('Servers/ManagedServer')\n");
buf.append("set('Machine', 'Machine_1')\n");
buf.append("set(ListenAddress, 7001)\n");
buf.append("save()\n");
buf.append("activate(block='true')\n");
```
package wlstest;
import java.io.IOException;
import java.lang.management.ManagementFactory;
import java.net.MalformedURLException;
import java.util.Hashtable;
import javax.management.MBeanServerConnection;
import javax.management.MalformedObjectNameException;
import javax.management.ObjectName;
import javax.management.remote.JMXConnector;
import javax.management.remote.JMXConnectorFactory;
import javax.management.remote.JMXServiceURL;
import javax.naming.Context;

public class JMXSample {
    private static MBeanServerConnection connection;
    private static JMXConnector connector;
    private static final ObjectName service;
    // Initializing the object name for DomainRuntimeServiceMBean
    // so it can be used throughout the class.
    static {
        try {
            service = new ObjectName("com.bea:Name=RuntimeService,Type=weblogic.management.runtime.ServerRuntimeMBean");
        } catch (MalformedObjectNameException e) {
            throw new AssertionError(e.getMessage());
        }
    }
    // Initialize connection to the Domain Runtime MBean Server
    public static void initConnection() throws IOException, MalformedURLException {
        String protocol = "t3";
        String hostname = "127.0.0.1";
        String portString = "7003";
        String username = "weblogic";
        String password = "welcome1";
        Integer portInteger = Integer.valueOf(portString);
        int port = portInteger.intValue();
        String jndiroot = "/jndi/";
        String mserver = "weblogic.management.mbeanservers.runtime";
        JMXServiceURL serviceURL = new JMXServiceURL(protocol, hostname, port, jndiroot + mserver);
        Hashtable h = new Hashtable();
        h.put(Context.SECURITY_PRINCIPAL, username);
        h.put(Context.SECURITY_CREDENTIALS, password);
        h.put(JMXConnectorFactory.PROTOCOL_PROVIDER_PACKAGES, "weblogic.management.remote");
        connector = JMXConnectorFactory.connect(serviceURL, h);
        connection = connector.getMBeanServerConnection();
    }
    public void getServerRuntimeInfo() throws Exception {
        System.out.printf("-------Get runtime info start----------\n");
        System.out.println("----------------OS Information----------------");
        // Get the number of processors
        int numProcessors = mxbean.getAvailableProcessors();
        System.out.printf("Processors:-- * numProcessors *=\n");
    }
}
System.out.println("*/");
System.out.println("System.out.println("----------Server Information---------");
ObjectName runtimeON = new ObjectName(  "com.bea:Name=RuntimeService,Type=weblogic.management.mbeanservers.runtime.RuntimeServiceMBean");
ObjectName server = (ObjectName) connection.getAttribute(runtimeON, "ServerConfiguration");
System.out.println("SERVER NAME:  " + connection.getAttribute(runtimeON, "ServerName"));
ObjectName domain = (ObjectName) connection.getAttribute(runtimeON, "DomainConfiguration");
System.out.println("DOMAIN NAME:  " + connection.getAttribute(domain, "Name"));

ObjectName runtime = (ObjectName) connection.getAttribute(runtimeON, "ServerRuntime");
//Get the cluster info
ObjectName cluster = (ObjectName) connection.getAttribute(runtime, "ClusterRuntime");
System.out.println("--------------Cluster Information------------");
System.out.println("Cluster name: " + connection.getAttribute(cluster, "Name"));
System.out.println("Alive Server Count: " + connection.getAttribute(cluster, "AliveServerCount"));
String[] servernames = (String[])connection.getAttribute(cluster, "ServerNames");
for (int i=0;i<servernames.length;++i)
System.out.printf(servernames[i] + " ");

//Get the JVM info
ObjectName jvm = (ObjectName) connection.getAttribute(runtime, "JVMRuntime");
System.out.println("--------------JVM Information------------");
System.out.println("Java Version:  " + connection.getAttribute(jvm, "JavaVersion"));
System.out.println("Java VMVendor:  " + connection.getAttribute(jvm, "JavaVMVendor"));
System.out.println("JVM HeapSize:  " + connection.getAttribute(jvm, "HeapSizeMax"));
System.out.println("Current Heap Size: " + connection.getAttribute(jvm, "HeapSizeCurrent"));
System.out.println("Heap Free Current: " + connection.getAttribute(jvm, "HeapFreeCurrent"));
System.out.println("Heap Free Percent: " + connection.getAttribute(jvm, "HeapFreePercent") + "% ");

//ObjectName runtimeJRockit = new ObjectName(  "com.bea:Name=RuntimeService,Type=weblogic.management.runtime.JRockitRuntimeMBean");
//ObjectName jrockit = (ObjectName) connection.getAttribute(runtime, "JRockitRuntime");
System.out.println("--------------JRockit Runtime Information------------");
System.out.println("OSVersion:  " + connection.getAttribute(jvm, "OSVersion"));
System.out.println("NumberOfProcessors:  " + connection.getAttribute(jvm, "NumberOfProcessors"));
System.out.println("AllProcessorsAverageLoad:  " + connection.getAttribute(jvm, "AllProcessorsAverageLoad"));
System.out.println("JVM Process load:  " + connection.getAttribute(jvm, "JvmProcessorLoad"));
System.out.println("TotalPhysicalMemory:  " + connection.getAttribute(jvm, "TotalPhysicalMemory"));
System.out.println("FreePhysicalMemory:  " + connection.getAttribute(jvm, "FreePhysicalMemory"));
System.out.println("UsedPhysicalMemory:  " + connection.getAttribute(jvm, "UsedPhysicalMemory"));
System.out.println("FreeHeap:  " + connection.getAttribute(jvm, "FreeHeap"));
System.out.println("FreeHeapPercent: " + connection.getAttribute(jvm, "FreeHeapPercent"));
System.out.println("TotalNumberOfThreads: " + connection.getAttribute(jvm, "TotalNumberOfThreads"));

//Get the Thread Pool info
ObjectName thread = (ObjectName) connection.getAttribute(runtime, "ThreadPoolRuntime");
System.out.println("-------------Thread Pool Information-------------");
System.out.println("Throughput:  " + connection.getAttribute(thread, "Throughput"));
System.out.println("ExecuteThreadIdleCount: " + connection.getAttribute(thread, "ExecuteThreadIdleCount"));
System.out.println("PendingUserRequestCount: " + connection.getAttribute(thread, "PendingUserRequestCount"));
System.out.println("HoggingThreadCount: " + connection.getAttribute(thread, "HoggingThreadCount"));
System.out.println("ExecuteThreadIdleCount: " + connection.getAttribute(thread, "ExecuteThreadIdleCount"));
System.out.println("PendingRequestOldestTime: " + connection.getAttribute(thread, "PendingRequestOldestTime"));
System.out.println("ExecuteThreadTotalCount: " + connection.getAttribute(thread, "ExecuteThreadTotalCount");

//Get the Execute Queue info
ObjectName execute = (ObjectName) connection.getAttribute(runtime, "DefaultExecuteQueueRuntime");
System.out.println("-------------Execute Queue Information-------------");
System.out.println("ExecuteThreadIdleCount: " + connection.getAttribute(execute, "ExecuteThreadIdleCount"));
System.out.println("HoggingThreadCount: " + connection.getAttribute(execute, "HoggingThreadCount"));
System.out.println("PendingUserRequestCount: " + connection.getAttribute(execute, "PendingUserRequestCount"));
System.out.println("PendingRequestOldestTime: " + connection.getAttribute(execute, "PendingRequestOldestTime"));
System.out.println("ExecuteThreadIdleCount: " + connection.getAttribute(execute, "ExecuteThreadIdleCount");
}
public static void main(String[] args) throws Exception {
    JMXSample s = new JMXSample();
    initConnection();
    //s.printNameAndState();
    s.getServerRuntimeInfo();
    connector.close();
}