

WebLogic Server 11gR1 Diagnostics Lab

Overview

The following hands-on labs are intended to provide an introduction to the WebLogic Server Diagnostic Framework (WLDF), a framework of diagnostic utilities built into the WebLogic Server architecture. The labs are intended to give you practice in configuring the major features of WLDF and to provide a template for configuring diagnostics for your own WebLogic Server domains.

There are 13 labs in all. These cover:

1. Browsing WebLogic Server MBeans
2. Changing WebLogic Server Debug settings
3. Configuring a JDBC Diagnostic Archive
4. Creating a System Diagnostic Module
5. System-Scoped Diagnostic Monitors (WLDF Profiles)
6. Enabling Application-Scoped Instrumentation
7. Exporting and Transforming WLDF Event Data
8. Configuring Collected Metrics Harvesting
9. Exporting and Transforming WLDF Collected Metrics Data
10. Creating a Diagnostic Watch
11. Configuring WLDF Notifications

The lab materials are included in a zipped archive file (DiagnosticLab.zip). You can unzip this and store it anywhere on your local machine, this folder will be referred below as %LAB_HOME%. In it, you will find the following folders:

Apps – Web apps used in the labs

Notifications – Utility classes and scripts for handling notifications

Profiles – Copies of WebLogic Server standard diagnostic profiles

Shortcuts – A number of useful MS-Windows shortcuts for starting servers etc.

SQL – SQL scripts for creating database schemas

Transforms – XSLT stylesheets for viewing WLDF archive data

Utilities – Useful diagnostic tools for WebLogic Server

Lab Setup – Setting Up the DiagnosticsLab Domain

Create a simple WebLogic Server 11gR1 domain using the Configuration Wizard. All you need for this lab is a simple domain with one server – in this guide we will assume that you have created a domain, with the following settings:

- Domain name: `DiagnosticsLab`
- Administrator credentials: `weblogic/weblogic1`
- Development mode, JRockit SDK
- AdminServer, listen port: 7001
- Do not configure managed servers, clusters and machines

When you have created the domain, before starting the AdminServer, we want to make a number of edits to `%DOMAIN_HOME%/bin/setDomainEnv.cmd`:

Use JRockit to enable Thread Dumps

Add the following line:

```
set JAVA_VENDOR=Oracle
```

near the beginning of the file, before the line:

```
if "%JAVA_VENDOR%"=="Oracle" (
```

Enable Fastswap to work with WLDF

Look for the line:

```
set enableHotswapFlag=
```

and change it this way:

```
set enableHotswapFlag=-javaagent:%WL_HOME%\server\lib\diagnostics-agent.jar
```

Disable wsee async response (BEA-22013) warnings

Add `-Dweblogic.wsee.skip.async.response=true` to the `JAVA_PROPERTIES` environment variable:

```
set JAVA_PROPERTIES=-Dplatform.home=%WL_HOME% -  
Dwls.home=%WLS_HOME% -Dweblogic.home=%WLS_HOME% -  
Dweblogic.wsee.skip.async.response=true
```

Modify the file paths for the shortcuts provided inside `%LAB_HOME%/Shortcuts`, these shortcuts will be used through all the lab exercises

Lab 1- Browsing WebLogic Server MBeans

In this exercise, you will practice browsing the WebLogic Server MBean trees. Start Diagnostics domain's AdminServer (you can use [%LAB_HOME%\Shortcuts\Admin Server.Ink](#))

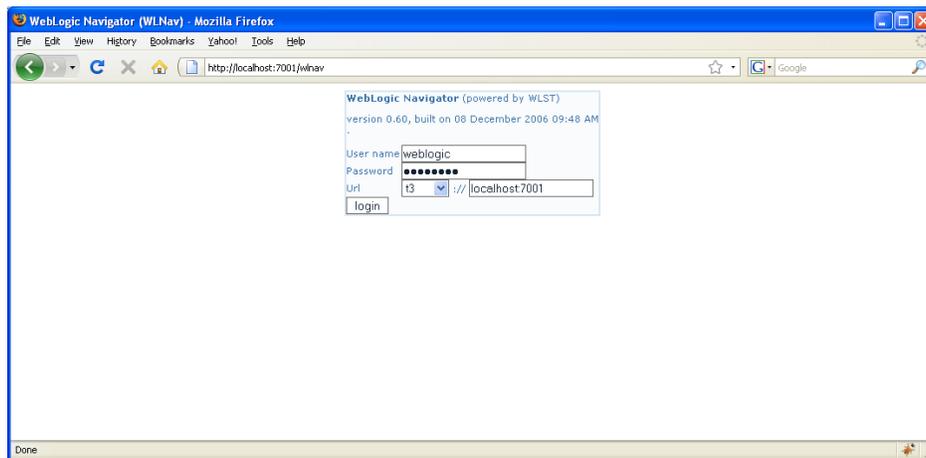
For this exercise, we will use both WLST and WLNav, an open-source utility originally developed by WebLogic Server consultants.. You will find a copy of WLNav in the %LAB_HOME%/Utilities folder.

WLNav is packaged as a web app archive (WLNav.war) and you can deploy it easily using either the admin console Deployments page or the following weblogic.Deployer command (don't forget to call setDomainEnv.cmd first):
`java weblogic.Deployer -username weblogic -password weblogic1 -targets AdminServer -deploy %LAB_HOME%\Utilities\wlnav.war`

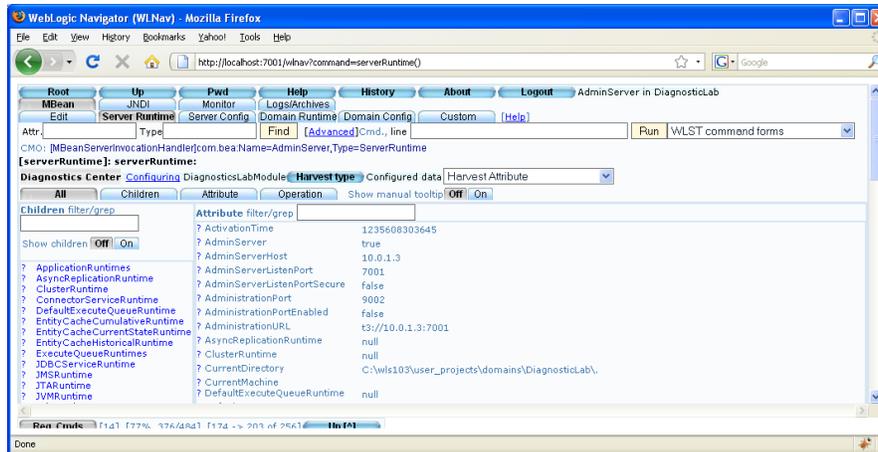
You should also deploy browsestore and shopping applications, located in %LAB_HOME%\apps
`java weblogic.Deployer -username weblogic -password weblogic1 -targets AdminServer -deploy %LAB_HOME%\Apps\Browsestore\app\browsestore.war`

`java weblogic.Deployer -username weblogic -password weblogic1 -targets AdminServer -deploy %LAB_HOME%\Apps\Shopping\app\ShoppingCart.war`

Browse to <http://localhost:7001/wlnav> and you should see something like this:

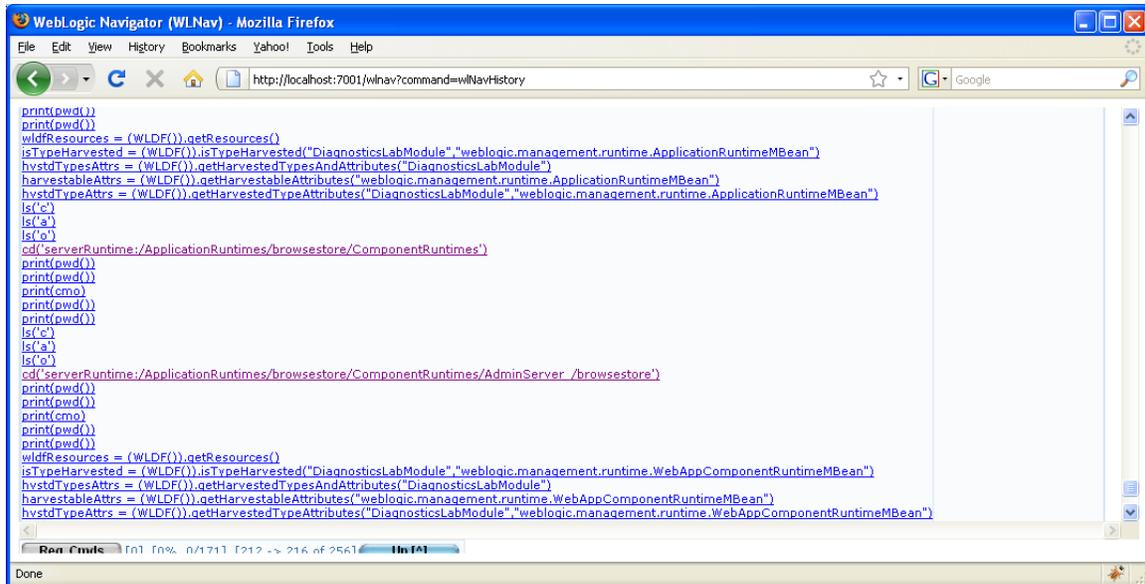


Enter the password for the DiagnosticsLab AdminServer ('weblogic1') and click the 'login' button. You should now see the main WLNav screen, which looks like this:



You can switch between the various MBean trees (e.g. Server Runtime/Config, Domain Runtime/Config) by clicking on the tabs at the top of the screen. Try selecting the **ServerRuntime** MBeans, then click on the **ApplicationRuntimes** link, then select an application (**browsestore.war**). Next click on **ComponentRuntimes** -> **AdminServer** -> **browsestore** to view runtime information about the instance of the browsestore.war application currently running on AdminServer. Next click on **Servlets->Welcome.jsp**, scroll down to view the Attributes in the pane on the right of the screen. For example, you can see the **InvocationTotalCount** attribute. Open a browser to <http://localhost:7001/browsestore>, refresh the WLN browser window and you will see **InvocationTotalCount** incremented by one.

Next, we will see how you can use weblogic.WLST to view the MBean tree in exactly the same way. We will take advantage of WLN's History tab: click on the tab at the top marked 'History' and then scroll all the way down to the bottom of the main pane. You should see a complete history of all the WLST commands that WLN has used to browse the MBean tree:



We will use the WLST commands (highlighted in purple) to navigate the MBean tree with WLST in online mode. Open a command shell, set your environment with `setDomainEnv.cmd` (or use the shortcut provided) and enter the following commands (you should see the WLST output as well, but I have left it out here for clarity):

```
>java weblogic.WLST
wls:/offline>connect ('weblogic', 'weblogic1', 't3://localhost:7001')
wls:/DiagnosticLab/serverConfig>cd ('serverRuntime:/ApplicationRuntimes')
wls:/DiagnosticLab/serverRuntime/ApplicationRuntimes>ls()
wls:/DiagnosticLab/serverRuntime/ApplicationRuntimes>cd('browsestore.war')
wls:/...> ls()
wls:/...> cd('ComponentRuntimes')
wls:/...> ls()
wls:/...> cd('AdminServer_/browsestore')
wls:/...> ls()
```

You should see all the attributes for the browsestore application running on the AdminServer displayed like this:

```
WebLogic Shell - java weblogic.WLST
r--- JSPDebug false
r--- JSPKeepGenerated false
r--- JSPPageCheckSecs 1
r--- JSPVerbose true
r--- LogFilename null
r--- LogRuntime null
r--- ModuleId /browstore
r--- ModuleURI browstore
r--- Name AdminServer_/browstore
r--- OpenSessionsCurrentCount 0
r--- OpenSessionsHighCount 1
r--- ServletReloadCheckSecs 1
r--- ServletSessionsMonitoringIds null
r--- SessionCookieComment null
r--- SessionCookieDomain null
r--- SessionCookieMaxAgeSecs -1
r--- SessionCookieName JSESSIONID
r--- SessionCookiePath /
r--- SessionIDLength 52
r--- SessionInvalidationIntervalSecs 60
r--- SessionMonitoringEnabled false
r--- SessionTimeoutSecs 300
r--- SessionsOpenedTotalCount 1
r--- SingleThreadedServletPoolSize 5
r--- SourceInfo browstore
r--- SpringRuntimeMBean null
r--- Status DEPLOYED
r--- Type WebAppComponentRuntime
r--- WebPubSubRuntime null
r--- deleteInvalidSessions Void :
r--- getKodPersistenceUnitRuntime WebLogicMBean : String(unitName)
r--- getMonitoringId String : String(sessionId)
r--- getServletSession WebLogicMBean : String(sessionID)
r--- getSessionLastAccessedTime Long : String(monitoringId)
r--- getSessionMaxInactiveInterval Long : String(monitoringId)
r--- invalidateServletSession Void : String(monitoringId)
r--- preDeregister Void :
wls:/DiagnosticLab/ServerRuntime/ApplicationRuntimes/browstore/ComponentRuntimes/AdminServer_/browstore> _
```

Experiment by browsing around the WebLogic Server MBean trees using WLNav and/or WLST: don't forget, if you have problems with the syntax for WLST, just use WLNav and look up the command history.

When you have finished, disconnect from WLST with the following commands:

```
wls:/...> disconnect()
wls:/offline> exit()
```

Lab 2 – Changing WebLogic Server Debug settings

Start the DiagnosticsLab [AdminServer](#) and logon to the admin console. Navigate to the [Environment -> Servers -> AdminServer](#) page and select the [Debug](#) tab. This screen allows you to set the various debug flags that will cause WebLogic Server to output detailed debug messages from its subsystems. You should see two nodes: 'default' and 'weblogic'. Click on the '+' sign to expand the 'weblogic' node and you should see a list of subsystems like this:

Settings for AdminServer

Configuration Protocols Logging **Debug** Monitoring Control Deployments Services Security Notes

Use this page to define debug settings for this server.

Debug settings for this Server

Enable Disable Clear Showing 1 to 2 of 2 Prev

<input type="checkbox"/> Debug Scopes and Attributes	State
<input type="checkbox"/> default	Disabled
<input type="checkbox"/> weblogic	Disabled
<input type="checkbox"/> application	Disabled
<input type="checkbox"/> classloader	Disabled
<input type="checkbox"/> cluster	Disabled
<input type="checkbox"/> connector	Disabled
<input type="checkbox"/> core	Disabled
<input type="checkbox"/> debug	Disabled
<input type="checkbox"/> default	Disabled
<input type="checkbox"/> deploy	Disabled
<input type="checkbox"/> descriptor	Disabled
<input type="checkbox"/> diagnostics	Disabled
<input type="checkbox"/> ejb	Disabled

You can use this page to change WebLogic Server's debug settings dynamically. You would usually start by enabling debugging at the subsystem level; however, this will typically produce a great deal of output and so you will usually want to turn on particular debug flags within a given subsystem while turning off debug at the subsystem level, once you have isolated the problem or area you wish to investigate.

For this lab, try turning on debugging for the 'diagnostics' subsystem as a whole. If you expand the 'diagnostics' node, you will see that this covers a whole set of debug options as shown in the following screenshot:

[-] diagnostics	Enabled
[+] accessor	Enabled (Inherited)
[+] archive	Enabled (Inherited)
[+] collections	Enabled (Inherited)
[+] context	Enabled (Inherited)
[+] harvester	Enabled (Inherited)
[+] image	Enabled (Inherited)
[+] instrumentation	Enabled
[+] lifecycle	Enabled (Inherited)
[+] logging	Enabled (Inherited)
[+] module	Enabled (Inherited)
[+] query	Enabled (Inherited)
[+] snmp	Enabled (Inherited)
[+] watch	Enabled (Inherited)

Here we have specifically enabled debug messages for the instrumentation module as well as turning on debugging for the diagnostics subsystem as a whole. This means that all the other modules also have debug enabled. As you will see, this produces a great deal of output and you may want to limit that by turning off the debug flag for the diagnostics subsystem as a whole. Try experimenting with other debug options: other subsystems of interest might be 'servlet', 'deploy' or 'application'.

To view the debug messages, select the 'Logging' tab (next to 'Debug'), scroll down and expand the 'Advanced' section. Change the [Minimum severity to log](#) to 'Debug' so that the debug messages appear in the server's log located in [%DOMAIN_HOME%\servers\AdminServer\logs\AdminServer.log](#). If you watch the log as you work your way through the lab, you will see all the various diagnostic messages.

Lab 3 – Configuring a JDBC Diagnostic Archive

In this lab, you will configure your local Oracle Database 10g XE instance to act as a WLDF Diagnostic Archive. If you haven't already done so, please install Oracle Database 10g XE on your local system, and run [OracleXETNSListener](#) and [OracleServiceXE](#) windows services.

- Go to <http://localhost:8080/apex/> , enter your `system` as username and systems's password.
- Navigate to [Administration->Database Users](#) and click [Create](#) button.
- create a user 'weblogic' with password 'weblogic', grant CONNECT and RESOURCE roles to it.

The screenshot shows the Oracle Database Express Edition interface. The browser address bar shows `localhost:8080/apex/f?p=4500:2611:1730012493257529::NO:RP,2611::`. The page title is "ORACLE Database Express Edition". The user is logged in as "SYSTEM". The breadcrumb navigation is "Home > Administration > Manage Database Users > Create Database User".

The "Create Database User" form contains the following fields and options:

- Username:**
- Password:**
- Confirm Password:**
- Expire Password:**
- Account Status:** (dropdown menu)
- Default Tablespace:** **USERS**
- Temporary Tablespace:** **TEMP**

The "User Privileges" section includes:

- Roles:** CONNECT, RESOURCE, DBA
- Direct Grant System Privileges:**
 - CREATE DATABASE LINK
 - CREATE MATERIALIZED VIEW
 - CREATE PROCEDURE
 - CREATE PUBLIC SYNONYM
 - CREATE ROLE
 - CREATE SEQUENCE
 - CREATE SYNONYM
 - CREATE TABLE
 - CREATE TRIGGER
 - CREATE TYPE
 - CREATE VIEW

At the bottom right of the "User Privileges" section, there are links for [Check All](#) and [Uncheck All](#).

- Open command prompt and run sqlplus with weblogic user's credentials. You should be logged in successfully.

```
C:\WINDOWS\system32\cmd.exe - sqlplus weblogic/weblogic@XE
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\dnefedki>sqlplus weblogic/weblogic@XE

SQL*Plus: Release 10.2.0.1.0 - Production on Tue Feb 8 11:49:49 2011
Copyright (c) 1982, 2005, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Express Edition Release 10.2.0.1.0 - Production
SQL> _
```

- Before creating the data source to use for the Diagnostic Archive, you need to create the database schema that WebLogic Server will use to store its diagnostic data. WebLogic Server uses two tables: [WLDF_HVST](#) (to store WLDF harvested metric data) and [WLS_EVENTS](#) (to store WLDF event data). A SQL script to create these tables is available in [%LAB_HOME%\SQL\WLDF_Data_Archive_Oracle.ddl](#). Execute this script from sqlplus to create the tables.

```
C:\WINDOWS\system32\cmd.exe - sqlplus weblogic/weblogic@XE
SQL>
SQL> @d:\work\technologies\WLS\workshops\wls_adoption\Instructor\Labs\Diagnostic
Lab\SQL\WLDF_Data_Archive_Oracle.ddl
DROP TABLE wls_events
*
ERROR at line 1:
ORA-00942: table or view does not exist

Table created.
DROP TABLE wls_hvst
*
ERROR at line 1:
ORA-00942: table or view does not exist

Table created.

Commit complete.
SQL> _
```

When you have created the database schema, use the admin console to create a new data source to use for the JDBC Diagnostic Archive.

- Navigate to the [Services-> Data Sources](#) page and select [New->Generic Data Source](#).
- Set [Name](#) = [ArchiveDS](#), [JNDI Name](#) = [ArchiveDS](#), [Database type](#) = [Oracle](#), click [Next](#)
- Use the default database driver, click [Next](#) twice

- Enter the following in the [Connection Properties](#) page: Database name = [XE](#), Host Name = [localhost](#), port = [1521](#), database username = [weblogic](#), password = [weblogic](#). Click [Next](#)
- Press [Test Configuration](#) button. You should see the message "[Connection Test Succeeded](#)". Click [Next](#).
- Select [AdminServer](#) as a target, click [Finish](#).

When you have created the JDBC data source (ArchiveDS), configure a JDBC Diagnostic Archive using the admin console:

- navigate to [Diagnostics -> Archives](#)
- select [AdminServer](#)
- set Type = [JDBC](#), Data Source = [ArchiveDS](#), click [Save](#).

Restart the AdminServer server to consume this configuration change.

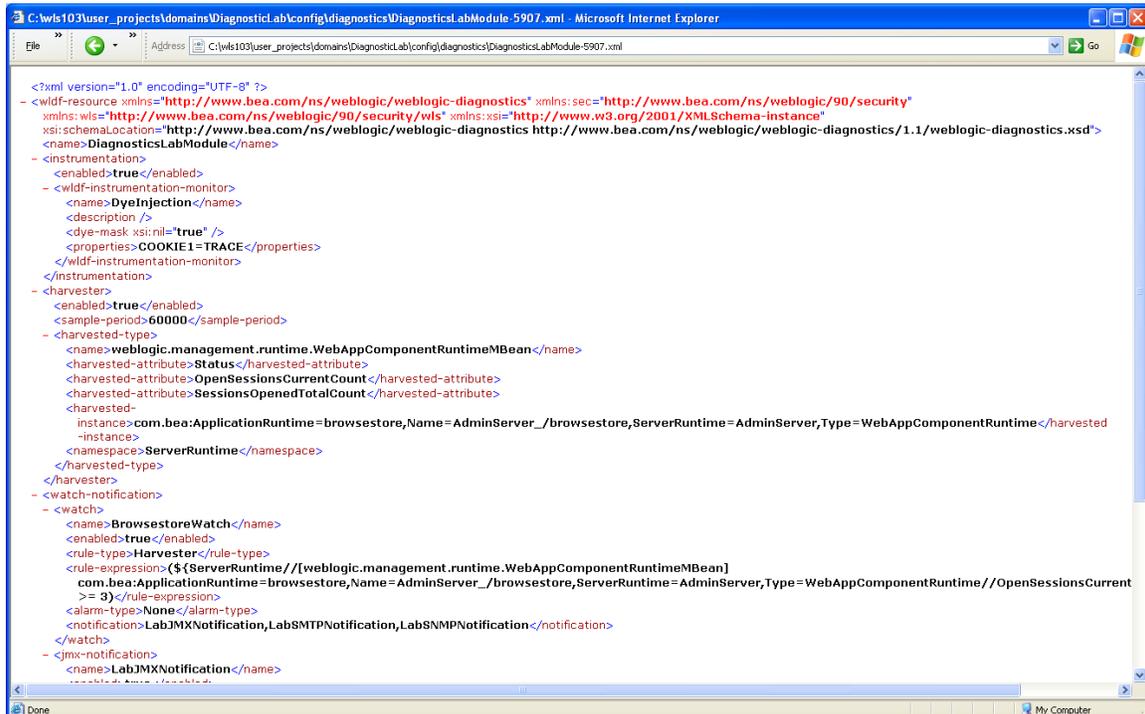
Lab 4 – Creating a System Diagnostic Module

To use the WebLogic Diagnostic Framework on AdminServer, you need to create a System Diagnostic Module.

- Log on to the admin console and go to [Diagnostics -> Diagnostic Modules](#).
- Create [New](#) button, name your new Diagnostic Module [DiagnosticLabModule](#) and click [Ok](#).
- Click on the newly created module and go to the [Targets](#) tab.
- Set AdminServer.as a target for the module, click [Save](#).

Notice that besides the 'General' tab, there are three additional tabs that you use to configure [Metrics](#); [Watches](#) and [Notifications](#); and [Instrumentation](#) at a server level. We will work with each of these in turn, but note that you can control the operation of the WebLogic Diagnostic Framework at the server level by enabling or disabling these features in the system diagnostic module.

To see how the configuration for the diagnostic subsystem is persisted, open a file browser and go to the `<domain_home>/config/diagnostics` folder to view the configuration files. There will be a file called `DiagnosticLabModule-<x>.xml` and if you open this you will see that it contains the configuration for the diagnostic subsystem. As you progress through the lab exercises, you may want to return and view this file to see how WebLogic Server stores its diagnostic configuration. Here is an example of a diagnostic system module with instrumentation, watch, notification and harvester elements configured.



```
<?xml version="1.0" encoding="UTF-8" ?>
- <wldf-resource xmlns="http://www.bea.com/ns/weblogic/weblogic-diagnostics" xmlns:sec="http://www.bea.com/ns/weblogic/90/security"
  xmlns:wls="http://www.bea.com/ns/weblogic/90/security/wls" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.bea.com/ns/weblogic/weblogic-diagnostics http://www.bea.com/ns/weblogic/weblogic-diagnostics/1.1/weblogic-diagnostics.xsd">
  <name>DiagnosticLabModule</name>
  - <instrumentation>
    <enabled>true</enabled>
    - <wldf-instrumentation-monitor>
      <name>DyeInjection</name>
      <description />
      <dye-mask xsi:nil="true" />
      <properties>COOKIE1=TRACE</properties>
    </wldf-instrumentation-monitor>
  </instrumentation>
  - <harvester>
    <enabled>true</enabled>
    <sample-period>60000</sample-period>
  - <harvested-type>
    <name>weblogic.management.runtime.WebAppComponentRuntimeMBean</name>
    <harvested-attribute>Status</harvested-attribute>
    <harvested-attribute>OpenSessionsCurrentCount</harvested-attribute>
    <harvested-attribute>SessionsOpenedTotalCount</harvested-attribute>
    <harvested-
      instance>com.bea:ApplicationRuntime=browsstore,Name=AdminServer_/browsstore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime</harvested
      -instance>
    <namespace>ServerRuntime</namespace>
  </harvested-type>
  </harvester>
  - <watch-notification>
    - <watch>
      <name>BrowsstoreWatch</name>
      <enabled>true</enabled>
      <rule-type>Harvester</rule-type>
      <rule-expression>({$ServerRuntime//[weblogic.management.runtime.WebAppComponentRuntimeMBean]
        com.bea:ApplicationRuntime=browsstore,Name=AdminServer_/browsstore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime//OpenSessionsCurrent
        >= 3}</rule-expression>
      <alarm-type>None</alarm-type>
      <notification>LabJMXNotification,LabSMTPNotification,LabSNMPNotification</notification>
    </watch>
    - <jmx-notification>
      <name>LabJMXNotification</name>
```

Lab 5 – System-Scoped Diagnostic Monitors (WLDF Profiles)

WebLogic Server 11gR1 ships with a number of example WLST scripts that you can use to configure diagnostic data gathering for most of the major subsystems.

These can be found with the WebLogic Server 11gR1 examples:

[%MIDDLEWARE_HOME%/wlserver_10.3/samples/server/examples/src/example_s/diagnostics/wldfprofiles/src](#)

You will also find a copy of these scripts in the [%LAB_HOME%/Profiles](#) folder, for use with the labs.

There are a number of Jython files with utility classes used to create, enable and disable WLDF diagnostic profiles as well as WLST scripts to create notifications using SMTP (email) and SNMP (network management) notifications. We will look at how to configure various notification methods in the following labs, so for now let's concentrate on the WLST scripts to enable/disable diagnostic metric and event data collection for the major WebLogic subsystems (Core, EJB, JDBC, JMS, JTA, WebApp and Web Services). There is a WLDF Profiles Overview document in the

[%MIDDLEWARE_HOME%/wlserver_10.3/samples/server/examples/src/example_s/diagnostics/wldfprofiles](#) directory ([Instructions.html](#)): this has more detail on the different profiles and how to use them. As you will see, It is very easy to use these diagnostic profiles with any WebLogic Server domain.

Open a command shell (set your environment with `setDomainEnv.cmd`), cd to [%LAB_HOME%/Profiles](#) and run the following `weblogic.WLST` command to enable all profiles on AdminServer:

```
>java weblogic.WLST enableAllProfiles.py user="weblogic" pass="weblogic1"
url="t3://localhost:7001" wldfResource="DiagnosticLabModule"
targets="AdminServer" harvesterPeriod=60000
```

You should see several screens of output messages from WLST, rather like this:

```

WebLogic Shell
Adding monitor DyeInjection
Adding weblogic.management.runtime.JVMRuntimeMBean to harvestables collection for DiagnosticsLabModule
Adding weblogic.management.runtime.WorkManagerRuntimeMBean to harvestables collection for DiagnosticsLabModule
Adding weblogic.management.runtime.JRockitRuntimeMBean to harvestables collection for DiagnosticsLabModule
Adding weblogic.management.runtime.ThreadPoolRuntimeMBean to harvestables collection for DiagnosticsLabModule
Adding weblogic.management.runtime.ServerRuntimeMBean to harvestables collection for DiagnosticsLabModule
Adding weblogic.management.runtime.ClusterRuntimeMBean to harvestables collection for DiagnosticsLabModule
Creating watch 'ServerLogWatch', type 'Log', with rule 'SEVERITY IN ('CRITICAL', 'ERROR', 'WARNING')', reset type: 'AutomaticReset', reset Per
iod: 120000
notifs: [IMBeanServerInvocationHandlerIcom.bea:Name=nyJMXNotif.Type=weblogic.diagnostics.descriptor.WLDFJMXNotificationBean.Parent=[Diagnost
icLab]/WLDFSystemResources [DiagnosticsLabModule],Path=WLDFResource [DiagnosticsLabModule]/WatchNotification [DiagnosticsLabModule]/JMXNotifica
tions [nyJMXNotif],IMBeanServerInvocationHandlerIcom.bea:Name=nySNMPPNotif.Type=weblogic.diagnostics.descriptor.WLDFSNMPPNotificationBean.Pare
nt=[DiagnosticsLab]/WLDFSystemResources [DiagnosticsLabModule],Path=WLDFResource [DiagnosticsLabModule]/WatchNotification [DiagnosticsLabModule]
/SNMPPNotifications [nySNMPPNotif]]
Creating watch 'JRockitCPUWatch', type 'Harvester', with rule '$([weblogic.management.runtime.JRockitRuntimeMBean]/AllProcessorsWatch) > 0.9',
reset type: 'AutomaticReset', reset Period: 120000
notifs: [IMBeanServerInvocationHandlerIcom.bea:Name=nyJMXNotif.Type=weblogic.diagnostics.descriptor.WLDFJMXNotificationBean.Parent=[Diagnost
icLab]/WLDFSystemResources [DiagnosticsLabModule],Path=WLDFResource [DiagnosticsLabModule]/WatchNotification [DiagnosticsLabModule]/JMXNotifica
tions [nyJMXNotif],IMBeanServerInvocationHandlerIcom.bea:Name=nySNMPPNotif.Type=weblogic.diagnostics.descriptor.WLDFSNMPPNotificationBean.Pare
nt=[DiagnosticsLab]/WLDFSystemResources [DiagnosticsLabModule],Path=WLDFResource [DiagnosticsLabModule]/WatchNotification [DiagnosticsLabModule]
/SNMPPNotifications [nySNMPPNotif]]
Creating watch 'HeapWatch', type 'Harvester', with rule '$([weblogic.management.runtime.JVMRuntimeMBean]/HeapFreePercent) < 10', reset type:
'AutomaticReset', reset Period: 120000
notifs: [IMBeanServerInvocationHandlerIcom.bea:Name=nyJMXNotif.Type=weblogic.diagnostics.descriptor.WLDFJMXNotificationBean.Parent=[Diagnost
icLab]/WLDFSystemResources [DiagnosticsLabModule],Path=WLDFResource [DiagnosticsLabModule]/WatchNotification [DiagnosticsLabModule]/JMXNotifica
tions [nyJMXNotif],IMBeanServerInvocationHandlerIcom.bea:Name=nySNMPPNotif.Type=weblogic.diagnostics.descriptor.WLDFSNMPPNotificationBean.Pare
nt=[DiagnosticsLab]/WLDFSystemResources [DiagnosticsLabModule],Path=WLDFResource [DiagnosticsLabModule]/WatchNotification [DiagnosticsLabModule]
/SNMPPNotifications [nySNMPPNotif]]
Creating watch 'ClusterServerWatch', type 'Harvester', with rule '$([weblogic.management.runtime.ClusterRuntimeMBean]/AliveServerCount) = 0',
reset type: 'AutomaticReset', reset Period: 120000
notifs: [IMBeanServerInvocationHandlerIcom.bea:Name=nyJMXNotif.Type=weblogic.diagnostics.descriptor.WLDFJMXNotificationBean.Parent=[Diagnost
icLab]/WLDFSystemResources [DiagnosticsLabModule],Path=WLDFResource [DiagnosticsLabModule]/WatchNotification [DiagnosticsLabModule]/JMXNotifica
tions [nyJMXNotif],IMBeanServerInvocationHandlerIcom.bea:Name=nySNMPPNotif.Type=weblogic.diagnostics.descriptor.WLDFSNMPPNotificationBean.Pare
nt=[DiagnosticsLab]/WLDFSystemResources [DiagnosticsLabModule],Path=WLDFResource [DiagnosticsLabModule]/WatchNotification [DiagnosticsLabModule]
/SNMPPNotifications [nySNMPPNotif]]
Creating watch 'StuckThreadWatch',

```

Now open the admin console and navigate to the [Diagnostics -> Diagnostic Modules](#) page; click on [DiagnosticLabModule](#) to view the new configuration and start by selecting the [Collected Metrics](#) tab. You will see a number of key WebLogic Server metrics are now being collected, like this:

Configuration | Targets

General | **Collected Metrics** | Watches and Notifications | Instrumentation

This section of the page lets you configure the enable metrics and sampling period

Enabled Enable metrics collection within this module. [More Info...](#)

Sampling Period: Sampling period in milliseconds. [More Info...](#)

This section of the page lets you configure collected metrics

[Customize this table](#)

Collected Metrics in this Module

Showing 1 to 10 of 25 Previous | Next

Metrics	Enabled
<input type="checkbox"/> Metrics	Enabled
<input type="checkbox"/> ClusterRuntimeMBean	true
<input type="checkbox"/> EJBCacheRuntimeMBean	true
<input type="checkbox"/> EJBLockingRuntimeMBean	true
<input type="checkbox"/> EJBPoolRuntimeMBean	true
<input type="checkbox"/> EJBTimerRuntimeMBean	true
<input type="checkbox"/> EJBTransactionRuntimeMBean	true
<input type="checkbox"/> JDBCDataSourceRuntimeMBean	true
<input type="checkbox"/> JMSConnectionRuntimeMBean	true
<input type="checkbox"/> JMSDestinationRuntimeMBean	true
<input type="checkbox"/> JMSPooledConnectionRuntimeMBean	true

Click on one of the metric sets (e.g. [JDBCDataSourceRuntimeMBean](#)) and notice that the system has been configured to collect metrics on all available attributes of this runtime MBean, which will provide a wealth of data about the health and performance of the JDBC subsystem:

Settings for JDBCDataSourceRuntimeMBean

General

Instances

Save

Use this page to configure an existing metric. A metric represents data from one or more attributes of the specified type.

Metric Type:

JDBCDataSourceRuntimeMBean

MBean Server location:

ServerRuntime

Enable Metric

Collected Attributes:

Available:

- ActiveConnectionsHighCount
- ConnectionsTotalCount
- CurrCapacityHighCount
- DatabaseProductName
- DatabaseProductVersion
- DeploymentState
- DriverName

Chosen:

- ActiveConnectionsAverage
- ActiveConnectionsCurrent
- ConnectionDelayTime
- CurrCapacity
- FailuresToReconnectCount
- LeakedConnectionCount
- NumAvailable

To collect JDBC metric data on the [ArchiveDS](#) data source you created earlier, all you now have to do is switch to the Instances tab and move the MBean instance from 'Available' to 'Chosen' and [Save](#). Remember to check that the Metric ('JDBCDataSourceRuntimeMBean') has been enabled and that metric collection is also enabled in the system diagnostic module.

If you now return to the [DiagnosticLabModule](#) page and select the [Watches and Notifications](#) tab, you will see that a number of WLDF watches have also been configured:

Watches Notifications

Use this page to add watches to the current diagnostic module and to configure those watches. Click the name of an existing watch to configure that watch.

[Customize this table](#)

Watches

New Delete Show all

<input type="checkbox"/>	Name ↕	Type	Enabled	Alarm Type
<input type="checkbox"/>	AvailableDSWatch	Collected Metrics	true	Automatic Reset
<input type="checkbox"/>	ClusterServerWatch	Collected Metrics	true	Automatic Reset
<input type="checkbox"/>	ConnectFailures	Collected Metrics	true	Manual Reset
<input type="checkbox"/>	ConnectionDelayWatch	Collected Metrics	true	Automatic Reset
<input type="checkbox"/>	HeapWatch	Collected Metrics	true	Automatic Reset
<input type="checkbox"/>	JRockitCPUWatch	Collected Metrics	true	Automatic Reset
<input type="checkbox"/>	JRockitHeapWatch	Collected Metrics	true	Automatic Reset
<input type="checkbox"/>	ServerHealthStateWatch	Collected Metrics	true	Automatic Reset
<input type="checkbox"/>	ServerLogWatch	Server Log	true	Automatic Reset
<input type="checkbox"/>	StuckThreadWatch	Collected Metrics	true	Automatic Reset

New Delete Show all

Click on one of these to view the configuration (for example, [JRockitHeapWatch](#)) and then select the Rule Expression tab to view the definition of the watch rule. In this case, it is:

```



$$\{\{weblogic.management.runtime.JRockitRuntimeMBean\}/\text{HeapFreePercent}\} < 10$$



```

This means that WLDF will watch for a condition where JRockit is reporting that less than 10% of the available heap memory is free. Remember that you can always enable/disable this particular watch, and you can also enable/disable all watches in the configuration for the system diagnostic module. The watch is now configured and you can easily create notifications that will fire if the watch rule evaluates to true (we will see how to do this later). The diagnostic profile script will already have created two notifications, using JMX and SNMP, and you now only have to complete the configuration for those notifications (for example, providing javamail properties for the SMTP notification).

You might like to revisit the system diagnostic module's configuration file `<domain_home>/config/diagnostics/DiagnosticLabModule-<x>.xml` to view the changes that have been added to the system diagnostic descriptor.

Lab 6 – Enabling Application-Scoped Instrumentation

One of the powerful features of the WebLogic Diagnostic Framework is its ability to support instrumentation of application code, using the AspectJ implementation of the Aspect-Oriented Programming (AOP) model. Essentially, this enables you to define various diagnostic actions (such as trace messages, display arguments and return values, elapsed time statistics, stack traces and thread dumps) that are to be executed whenever any method matching a particular pattern (called a 'pointcut') is called or executed. There is no need to make any modifications to the application source code and in fact instrumentation can be enabled on an application deployed as a packaged Java archive (such as an .ear or .war file).

WebLogic Server provides an abstraction layer to simplify the task of configuring application instrumentation, using constructs called diagnostic monitors which are pre-configured to execute certain diagnostic actions, leaving the programmer or administrator to specify only the context in which diagnostic traces are wanted. It is possible to configure system-scoped diagnostic monitors that will execute, for example, whenever any servlet session is created, or whenever a JDBC connection is started. This makes it easy to configure application instrumentation but often these system-scoped diagnostic monitors produce too much diagnostic data, particularly in a large and complicated application. In such cases, it is often better to use application-scoped instrumentation, which allows more fine-grained control over the diagnostic data that is generated.

In this lab, you will, learn how to configure application-scoped instrumentation for custom applications. We will work with the [browsestore](#) and [ShoppingCart](#) web applications – these are simple webapps that contain a number of Java classes that implement the servlet interface. We will configure instrumentation to gather diagnostic data on these servlet classes.

You will find a copy of the browsestore application in the `%LAB_HOME%/apps` folder. You will find a folder called [browsestore](#) with two sub-folders, [app](#) and [plan](#): the first contains [browsestore.war](#) in exploded .war format and the second will contain the Deployment Plan, which we will use to reconfigure the diagnostic module later on. We will not be versioning the app, so this directory structure will suffice for now.

Remember that you configured the domain to support Fastswap with the WebLogic Diagnostic Framework: this means that changes to compiled Java classes automatically reloaded via Fastswap can contain instrumentation – 'WebLogic Server' will invoke the AspectJ compiler to 'weave' the instrumentation into the newly-compiled classes.

First note that we have to enable Fastswap in the app's WEB-INF/weblogic.xml deployment descriptor, as follows:

```
<?xml version='1.0' encoding='UTF-8'?>
<weblogic-web-app xmlns="http://www.bea.com/ns/weblogic/90"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <fast-swap/>
</weblogic-web-app>
```

To enable instrumentation, an additional deployment descriptor is added to the app's META-INF directory, called weblogic-diagnostics.xml. Open this file with a browser or XML editor and look at the <instrumentation> section, which defines the application-scoped instrumentation for this application:

```
<instrumentation>

  <enabled>true</enabled>
  <include>com.servlets.*</include>

  <wldf-instrumentation-monitor>
    <name>ServletArgs</name>
    <enabled>true</enabled>
    <action>DisplayArgumentsAction</action>
    <location-type>before</location-type>
    <pointcut>call( * com.servlets.* get*(...));</pointcut>
  </wldf-instrumentation-monitor>

  <wldf-instrumentation-monitor>
    <name>ServletRetVal</name>
    <enabled>true</enabled>
    <action>DisplayArgumentsAction</action>
    <location-type>after</location-type>
    <pointcut>call( * com.servlets.* get*(...));</pointcut>
  </wldf-instrumentation-monitor>

  <wldf-instrumentation-monitor>
    <name>ServletElapsed</name>
    <enabled>true</enabled>
    <action>TraceElapsedTimeAction</action>
    <location-type>around</location-type>
    <pointcut>call( * com.servlets.* get*(...));</pointcut>
  </wldf-instrumentation-monitor>

  <wldf-instrumentation-monitor>
    <name>ServletStack</name>
    <enabled>true</enabled>
    <action>StackDumpAction</action>
    <location-type>before</location-type>
```

```

    <pointcut>call( * com.servlets.* getPaper*(...));</pointcut>
</wldf-instrumentation-monitor>

```

```

<wldf-instrumentation-monitor>
  <name>ServletThreads</name>
  <enabled>true</enabled>
  <action>ThreadDumpAction</action>
  <location-type>before</location-type>
  <pointcut>call( * com.servlets.* getFurn*(...));</pointcut>
</wldf-instrumentation-monitor>

```

```

<wldf-instrumentation-monitor>
  <name>ServletStats</name>
  <enabled>true</enabled>
  <action>MethodInvocationStatisticsAction</action>
  <location-type>around</location-type>
  <pointcut>call( * com.servlets.* get*(...));</pointcut>
</wldf-instrumentation-monitor>

```

```

</instrumentation>

```

The `<wldf-instrumentation>` stanzas configure diagnostic monitors that will produce a variety of diagnostic actions (e.g. [MethodInvocationStatisticsAction](#)) whenever any class which matches a certain pattern (or “pointcut”) is called. Of course, the secret to producing the most useful diagnostic data is to specify the classes to trace as tightly as possible.

To see the diagnostic monitors in operation, deploy the application using the admin console.

When you have done so, browse to the [Deployments -> browsestore -> Configuration -> Instrumentation](#) page and note that the diagnostic configuration from the `weblogic-diagnostics.xml` descriptor is reflected there:

Diagnostic Monitors in this Module

Add Monitor From Library Add Custom Monitor Remove Showing 1 to 6 of 6 Previous | Next

<input type="checkbox"/>	Name ↕	Type	Enabled	Actions
<input type="checkbox"/>	ServletArgs	Custom	true	DisplayArgumentsAction
<input type="checkbox"/>	ServletElapsed	Custom	true	TraceElapsedTimeAction
<input type="checkbox"/>	ServletRetVal	Custom	true	DisplayArgumentsAction
<input type="checkbox"/>	ServletStack	Custom	true	StackDumpAction
<input type="checkbox"/>	ServletStats	Custom	true	MethodInvocationStatisticsAction
<input type="checkbox"/>	ServletThreads	Custom	true	ThreadDumpAction

Add Monitor From Library Add Custom Monitor Remove Showing 1 to 6 of 6 Previous | Next

Make sure that your system diagnostic module ([DiagnosticLabModule](#)) has instrumentation enabled, check it in the [Diagnostics -> Diagnostics Modules -> DiagnosticLabModule -> Instrumentation Tab](#).

Settings for DiagnosticLabModule

Configuration Targets

General Collected Metrics Watches and Notifications **Instrumentation**

Save

Use this page to enable instrumentation and to define diagnostic monitors for this diagnostic module. A diagnostic monitor executes diagnostic actions when a specific location in the server or application example, a diagnostic monitor in a diagnostic system module can cause a trace event to be generated after a resource adapter has completed a transaction.

Enabled Enables instrumentation for this module. [More Info...](#)

Save

Run the application (<http://localhost:7001/browsestore>) and browse the store to generate some diagnostic trace information. You can see the diagnostic data (in raw form – we will look later at how to extract and refactor that data) by opening a SQL*Plus command window as weblogic user and running the following SQL query:

```
SQL> select monitor, classname, methodname from wls_events;
```

Because you will be generating a fair amount of diagnostic event data, you may find it easier to see the diagnostic data being generated if you first truncate the wls_events table (“SQL> truncate table wls_events; commit;”) before running the application; you can then view the fresh diagnostic traces by running the SQL select statement.

Lab 7 – Exporting and Transforming WLDF Event Data

Open a command shell and set your environment with `setDomainEnv.cmd`, CD to your `%LAB_HOME%\Transforms` directory. Use WLST to export your diagnostic event archive to a local XML file:

```
>java weblogic.WLST
wls:/offline> connect('weblogic','weblogic1','t3://localhost:7001')
wls:/DiagnosticLab/serverConfig>
exportDiagnosticDataFromServer(logicalName='EventsDataArchive',exportFileName='DiagnosticEvents.xml')
wls:/DiagnosticLab/serverConfig> disconnect()
wls:/offline> exit()
```

Open `DiagnosticEvents.xml` with a text editor. You will see that the XML file consists of a `<DataInfo>` stanza that describes the data that is held in this diagnostic archive: note how this matches the database schema you can view using SQL*Plus with the command: `'desc wls_events;'`. The rest of the XML file consists of multiple `<DataRecord>` stanzas which contain the actual event data. If you scroll down, you will see the payloads for the `StackDumpAction` and `ThreadDumpAction` events. Add the following Processing Instruction to use the `wldfEvents.xsl` Stylesheet provided (in the `DiagnosticsLab/Transforms` folder):

```
<?xml-stylesheet type="text/xsl" href="wldfEvents.xsl"?>
```

The start of the file should now look like this:

```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/xsl" href="wldfEvents.xsl"?>

<DiagnosticData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/ns/weblogic/90/diagnostics/accessor/export.xsd export.xsd"
xmlns="http://www.bea.com/ns/weblogic/90/diagnostics/accessor/Export">
  <DataInfo>
    ...
```

Open the file with a browser and you should see something like this:

Monitor	Class	Method	Payload
ServletElapsed	com.servlets.browseCategories	getPaperSupplies	21292092
ServletRetVal	com.servlets.browseCategories	getPaperSupplies	
ServletElapsed	com.servlets.browseCategories	getPaperSupplies	57486915
ServletRetVal	com.servlets.browseCategories	getPaperSupplies	
ServletArgs	com.servlets.browseCategories	getFurnitureSupplies	
ServletElapsed	com.servlets.browseCategories	getFurnitureSupplies	
<pre> ===== FULL THREAD DUMP ===== Wed Feb 11 10:20:54 2009 BEA JRockit(R) R27.6.0-50_o-100423-1.6.0_05- 20080626-2105-windows-ia32 "Main Thread" id=1 idx=0x4 tid=2780 prio=5 alive, in native, waiting -- Waiting for notification on weblogic3/srvr/T3Srvr@0x0B078FC8[fat lock] at jrockit/vm/Threads.waitForNonfySignal(JLjava/lang/Object;)Z(Native Method) at java/lang/Object.wait(J)V(Native Method) at java/lang/Object.wait(Object java:485) at weblogic3/srvr/T3Srvr.waitForDeath (T3Srvr.java:811) ^-- Lock released while waiting weblogic3/srvr/T3Srvr@0x0B078FC8[fat lock] at weblogic3/srvr/T3Srvr.run (T3Srvr.java:459) at weblogic/Server.man(Server.java:67) at jrockit/vm/RNI.c2java(IIII)V(Native Method) -- end of trace "(Signal Handler)" id=2 idx=0x8 tid=3652 prio=5 alive, in native, daemon "(GC Main Thread)" id=3 idx=0xc tid=4896 prio=5 alive, in native, native_waiting, daemon "(GC Worker Thread 1)" id=7 idx=0x10 tid=2488 prio=5 alive, in native, daemon "(GC Worker Thread 2)" id=7 idx=0x14 tid=4796 prio=5 alive, in native, daemon "(Code Generation Thread 1)" id=4 idx=0x18 tid=3904 prio=5 alive, in native, native_waiting, daemon "(Code Optimization Thread 1)" id=5 idx=0x1c tid=5296 prio=5 alive, in native, native_waiting, daemon "(VM Periodic Task)" id=6 idx=0x20 tid=5764 prio=10 alive, in native, daemon "(Attach Listener)" id=7 idx=0x24 tid=2760 prio=5 alive, in native, daemon "Finalizer" id=8 idx=0x28 tid=4420 prio=8 alive, in native, native_waiting, daemon at jrockit/memory/Finalizer.waitForFinalizes(Ljava/lang/Object;)I(Native Method) at jrockit/memory/Finalizer.access\$500(Finalizer.java:12) at jrockit/memory/Finalizer\$4.run(Finalizer.java:159) at java/lang/Thread.run(Thread.java:619) at jrockit/vm/RNI.c2java(IIII)V(Native Method) -- end of trace "Reference Handler" id=9 idx=0x2c tid=2252 prio=10 alive, in native, native_waiting, daemon at java/lang/ref/Reference.waitForActivatedQueue(Ljava/lang/ref/Reference,(Native Method) at java/lang/ref/Reference.access\$100 (Reference.java:11) at java/lang/ref/Reference\$ReferenceHandler.run(Reference.java:79) at jrockit/vm/RNI.c2java(IIII)V(Native Method) -- end of trace "(Sensor Event Thread)" id=10 idx=0x30 tid=4828 prio=5 alive, in native, daemon "Timer-0" id=13 idx=0x34 tid=4880 prio=5 alive, in native, waiting, daemon -- Waiting for notification on java/util/TaskQueue@0x0B078FC8[fat lock] at </pre>			

Have a quick look at the supplied [wldf_events.xsl](#) stylesheet, which you can customize if you wish to provide a custom view of the WLDF Event data. If you are familiar with XSLT transforms, you might be interested to look at Phil Aston's excellent article "Mining WebLogic Diagnostic Data with XSLT", which discusses how you can use XSLT to generate hierarchical reports from WLDF diagnostic events:

<http://www.oracle.com/technetwork/articles/entarch/mining-wldf-xslt-083813.html>

Lab 8 – Configuring Collected Metrics Harvesting

Open Admin Console , navigate to Diagnostics-> Diagnostics Modules -> [DiagnosticLabModule->Collected Metrics](#). Set the Sampling Period to 60000 ms and check that metrics is enabled

Settings for DiagnosticLabModule

Configuration Targets

General **Collected Metrics** Watches and Notifications Instrumentation

Save

This section of the page lets you configure the enable metrics and sampling period

Enabled

Sampling Period:

Save

Look through Collected Metrics table and try to find [WebAppComponentRuntimeMBean](#), If this metric exists, select it and click [Delete](#) button.

<input type="checkbox"/>	ServletRuntimeMBean
<input type="checkbox"/>	ThreadPoolRuntimeMBean
<input checked="" type="checkbox"/>	WebAppComponentRuntimeMBean
<input type="checkbox"/>	WorkManagerRuntimeMBean
<input type="checkbox"/>	WseeOperationRuntimeMBean

New Delete

Create a Metric based on the ServerRuntimeMBean server:

- Click [New](#)
- For the [MBean Server Location](#) choose [ServerRuntime](#)
- For the [MBean type](#), select custom [MBean Type](#) and enter: [weblogic.management.runtime.WebAppComponentRuntimeMBean](#)
- For Collected Attributes, choose: [Status](#), [OpenSessionsCurrentCount](#), [SessionsOpenedTotalCount](#)
- For Collected Instances, use: [com.bea:ApplicationRuntime=Shopping,Name=AdminServer_/ShoppingCart,ServerRuntime=AdminServer,Type=WebAppComponentRuntime](#)
- Click [Finish](#)

Open a few browser windows and run the ShoppingCart application (<http://localhost:7001/ShoppingCart>). To make sure our watch rule evaluates to

true, we want each browser window to open a new session to the application server. You can force this behaviour by clearing the browser's session cookies (Firefox: [Tools -> Options -> Privacy->Remove Individual Cookies](#), Internet Explorer: [Tools -> Internet Options -> General -> Delete Cookies](#)) and reloading the application.

Use the admin console to monitor the number of sessions by navigating to [Deployments -> browsestore -> monitoring -> Sessions](#). After a couple of minutes, you should be able to use SQL*Plus to view the WLDF Collected Metrics with the following SQL command:

```
SQL> select server, type, name, attrname, attrvalue from wls_hvst;
```

Lab 9 – Exporting and Transforming WLDF Collected Metrics Data

Open a command shell and set your environment with `setDomainEnv.cmd`, CD to your `%LAB_HOME%\Transforms` directory. Use WLST to export your diagnostic collected metrics archive to a local XML file:

```
>java weblogic.WLST
wls:/offline> connect('weblogic','weblogic1','t3://localhost:7001')
wls:/DiagnosticLab/serverConfig>
exportDiagnosticDataFromServer(logicalName='HarvestedDataArchive',exportFile
eName='DiagnosticMetrics.xml')
wls:/DiagnosticLab/serverConfig> disconnect()
wls:/offline> exit()
```

Open `DiagnosticMetrics.xml` with an editor and add the following processing instruction to use the `wldfMetrics.xsl` Stylesheet provided (in the `%LAB_HOME%\Transforms` folder):

```
<?xml-stylesheet type="text/xsl" href="wldfMetrics.xsl"?>
```

The start of the file should look like this:

```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/xsl" href="wldfMetrics.xsl"?>

<DiagnosticData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/ns/weblogic/90/diagnostics/accessor/e
xport.xsd export.xsd"
xmlns="http://www.bea.com/ns/weblogic/90/diagnostics/accessor/Export">
  <DataInfo>
```

...
Now open the file `DiagnosticMetrics.xml` with a browser and you should see something like this:

C:\abs\DiagnosticLab\DiagnosticMetrics.xml - Microsoft Internet Explorer

Address C:\abs\DiagnosticLab\DiagnosticMetrics.xml

Name	Attribute Name	Attribute Type	Attribute Value
com.bea.ApplicationRuntime=browsestore,Name=AdminServer_browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime	Status	5	DEPLOYED
com.bea.ApplicationRuntime=browsestore,Name=AdminServer_browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime	SessionsOpenedTotalCount	4	6
com.bea.ApplicationRuntime=browsestore,Name=AdminServer_browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime	SessionsOpenedTotalCount	4	1
com.bea.ApplicationRuntime=browsestore,Name=AdminServer_browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime	Status	5	DEPLOYED
com.bea.ApplicationRuntime=browsestore,Name=AdminServer_browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime	OpenSessionsCurrentCount	4	1
com.bea.ApplicationRuntime=browsestore,Name=AdminServer_browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime	SessionsOpenedTotalCount	4	1
com.bea.ApplicationRuntime=browsestore,Name=AdminServer_browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime	Status	5	DEPLOYED
com.bea.ApplicationRuntime=browsestore,Name=AdminServer_browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime	OpenSessionsCurrentCount	4	3
com.bea.ApplicationRuntime=browsestore,Name=AdminServer_browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime	SessionsOpenedTotalCount	4	3
com.bea.ApplicationRuntime=browsestore,Name=AdminServer_browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime	Status	5	DEPLOYED
com.bea.ApplicationRuntime=browsestore,Name=AdminServer_browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime	OpenSessionsCurrentCount	4	3
com.bea.ApplicationRuntime=browsestore,Name=AdminServer_browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime	SessionsOpenedTotalCount	4	3

Done My Computer

Lab 10 – Configuring Diagnostic Watches

In this lab, you will use the Admin Console to create a diagnostic watch:
Navigate to [Diagnostics](#) -> [Diagnostic Modules](#) -> [DiagnosticLabModule](#) -> [Watches and Notifications](#).

- Select the Watches tab and click [New](#):
- Set watch Name: [ShoppingWatch](#), watch Type: [Collected Metrics](#), Enable Watch: [Checked](#), click [Next](#)
- Click [Add Expressions](#) Button
- Select MBean Server Location: [ServerRuntime](#)
- Mbean Type:
[weblogic.management.runtime.WebAppComponentRuntimeMBean](#)
- Instance:
[com.bea:ApplicationRuntime=Shopping,Name=AdminServer_/ShoppingCard,ServerRuntime=AdminServer,Type=WebAppComponentRuntime](#)
- Message Attribute: [OpenSessionsCurrentCount](#), Operator: [>=](#) , Value: [3](#)
This will produce a watch rule:
([\\${ServerRuntime//\[weblogic.management.runtime.WebAppComponentRuntimeMBean\]com.bea:ApplicationRuntime=Shopping,Name=AdminServer_/ShoppingCart,ServerRuntime=AdminServer,Type=WebAppComponentRuntime//OpenSessionsCurrentCount} >= 3](#))
- Click [Finish](#) to go to the Configure Watch page – you could add additional watch expressions here and combine them to form a compound watch rule, but for now we'll keep it simple. Click on [Finish](#) to create the diagnostic watch.

At this point, you are able to configure an alarm (which enables you to control how frequently the rule condition will be re-evaluated): you would use the Alarm tab to do so, but we won't do so here. You can also configure Notifications that you want to fire when the watch rule expression evaluates to true, for example generating an SNMP trap, sending an email to the system administrator, putting a message onto a JMS queue or broadcasting a JMX notification. We will set up a set of notifications in the next lab. After configuring the notifications, you will hook the watch up to the notifications and test to see these in action.

Lab 11 – Configuring WLDf Notifications

Configuring an SNMP Trap Notification

In this lab, you will configure WebLogic Server to generate SNMP traps that can be monitored by network monitoring tools (such as HP Openview or IBM Tivoli). Open the admin console and navigate to [Diagnostics -> SNMP -> Agents](#) and create a new server-scoped SNMP agent as follows:

- Create a new Server SNMP Agent, name it [DiagnosticSNMPAgent](#)
- Select newly created [DiagnosticSNMPAgent](#)
- [Configuration->General](#) tab: SNMP UDP Port: [161](#), Enabled: [checked](#). Click [Save](#)
- [Targets](#) tab: set [AdminServer](#) as a target, click [Save](#).
- Go to [Configuration->Trap Destination](#), click [New](#)
Enter [DiagnosticTrapDestination](#) as a name, Community: public, Host: localhost, Port: 162. Click OK.
-

Now we need to create a new diagnostic notification that uses the SNMP trap destination.

- Navigate to the admin console [Diagnostics -> Diagnostic Modules -> DiagnosticLabModule](#) page and select the [Watches and Notification](#) tab
- Then Select Notifications and click on [New](#) to create a notification with the following properties:
- Type: SNMP Trap, Name: LabSNMPNotification, Enabled: checked

Configuring a JMX Notification

In this lab you will configure a JMX notification.

Navigate to the admin console [Diagnostics -> Diagnostic Modules -> DiagnosticLabModule](#) page and select the [Watches and Notification](#) tab. Then Select [Notifications](#) and click on [New](#) to create a notification with the following properties:

Name: LabJMXNotification
Type: JMX
Enabled: checked

Receiving SNMP/JMX Notifications

In this lab, you will test the diagnostic watch and notifications configured in the preceding labs. We have already created a watch ([ShoppingWatch](#)) but we need to hook it up to the newly-created notifications. Navigate to [Diagnostics -> Diagnostic Modules -> LabDiagnosticModule -> Watches and Notifications -> Watches -> ShoppingWatch -> Notifications](#) and move [LabSNMPNotification](#) and [LabJMXNotification](#) from Available to Chosen.

To view broadcast JMX notifications you need to run a custom JMX listener class. There is one provided in the %LAB_HOME%/Notifications folder along with a simple command script to run it. **You should correct the file paths in the %LAB_HOME%\Notifications\JMXListener.cmd** There is a also a Windows shortcut in the %LAB_HOME%/Shortcuts folder. Run the JMXListener.

To view SNMP trap notifications, you need to run the WebLogic Server SNMP commandline utility: `java weblogic.diagnostics.snmp.cmdline.Manager`. There is a simple command script to do this in the %LAB_HOME%/Notifications folder. **You should correct the file paths in the %LAB_HOME%\Notifications\SNMPListener.cmd** There is also another shortcut in the DiagnosticLab/Shortcuts folder.

Start three new browser windows (clear private data to ensure servlet sessions are not shared) and run ShoppingCart: <http://localhost:7001/ShoppingCart> . Use the admin console (Deployments -> Shopping -> Monitoring -> Sessions) to check that there are at least three sessions active:

Settings for Shopping

Overview Deployment Plan Configuration Security Targets Control Testing **Monitoring** Notes

Web Applications Servlets **Sessions** PageFlows Workload Web Service Clients

Use this page to view statistics about the sessions associated with this Web application.

Customize this table

Servlet Sessions (Filtered - More Columns Exist)

Context Root	Server	Creation Time	Time Last Accessed	Max Inactive Interval
/ShoppingCart	AdminServer	2/8/11 4:06:21 PM MSK	2/8/11 4:06:26 PM MSK	300
/ShoppingCart	AdminServer	2/8/11 4:06:14 PM MSK	2/8/11 4:06:18 PM MSK	300
/ShoppingCart	AdminServer	2/8/11 4:06:32 PM MSK	2/8/11 4:06:35 PM MSK	300

Wait a short time for the Watch to be evaluated and you should then see the Watch notifications via JMX, SNMP

```

C:\wls103\wlsesvr_10.3\samples\donains\wl_server>java weblogic.diagnostics.snmp.cmdline.Manager SnmpTrapMonitor -p 162
Listening on port:162
---
Snmp Trap Received ---
Version      : 01
Source       : udpEntity:127.0.0.1:163
Community    : public
Enterprise   : enterprises.140.625
TrapOID      : enterprises.140.625.0.85
RawTrapOID   : 1.3.6.1.4.1.140.625.0.85
Trap Objects :
< enterprises.140.625.100.5=Feb 11, 2009 8:54:39 PM PST >
< enterprises.140.625.100.145=DiagnosticLab >
< enterprises.140.625.100.10=AdminServer >
< enterprises.140.625.100.120=Notice >
< enterprises.140.625.100.105=BrowseStoreWatch >
< enterprises.140.625.100.110=Harvester >
< enterprises.140.625.100.115=<ServerRuntime//weblogic.management.runtime.WebAppComponentRuntimeMBeanIcon.bea:ApplicationRuntime-browsestore.Name=AdminServer/BrowseStore.ServerRuntime=AdminServer.Type=WebAppComponentRuntime//OpenSessionsCurrentCount >= 3 >
< enterprises.140.625.100.125=ServerRuntime//weblogic.management.runtime.WebAppComponentRuntimeMBeanIcon.bea:ApplicationRuntime-browsestore.Name=AdminServer/BrowseStore.ServerRuntime=AdminServer.Type=WebAppComponentRuntime//OpenSessionsCurrentCount >= 3 >
< enterprises.140.625.100.130=None >
< enterprises.140.625.100.135=60000 >
< enterprises.140.625.100.140=LabSNMPNotification >
Raw UarBinds :
< enterprises.140.625.100.5=Feb 11, 2009 8:54:39 PM PST >
< enterprises.140.625.100.145=DiagnosticLab >
< enterprises.140.625.100.10=AdminServer >
< enterprises.140.625.100.120=Notice >
< enterprises.140.625.100.105=BrowseStoreWatch >
< enterprises.140.625.100.110=Harvester >
< enterprises.140.625.100.115=<ServerRuntime//weblogic.management.runtime.WebAppComponentRuntimeMBeanIcon.bea:ApplicationRuntime-browsestore.Name=AdminServer/BrowseStore.ServerRuntime=AdminServer.Type=WebAppComponentRuntime//OpenSessionsCurrentCount >= 3 >
< enterprises.140.625.100.125=ServerRuntime//weblogic.management.runtime.WebAppComponentRuntimeMBeanIcon.bea:ApplicationRuntime-browsestore.Name=AdminServer/BrowseStore.ServerRuntime=AdminServer.Type=WebAppComponentRuntime//OpenSessionsCurrentCount >= 3 >
< enterprises.140.625.100.130=None >
< enterprises.140.625.100.135=60000 >
< enterprises.140.625.100.140=LabSNMPNotification >

```

```
WebLogic Shell - java_JMXWatchNotificationListener
C:\labs\DiagnosticLab>java JMXWatchNotificationListener
Adding shutdown hook
URL=service:jmx:t3://localhost:7001/jndi/weblogic.management.mbeanservers.runtime
Adding notification handler for: com.bea.Name-DiagnosticsJMXNotificationSource,ServerRuntime=AdminServer,Type=WLDFWatchJMXNotificationRuntime,WLDFRuntime=WLDFRuntime,WLDFWatchNotificationRuntime=WatchNotification
=====
Notification name: myjmx called. Count= 1.
Match severity: Notice
Match time: Feb 11, 2009 9:16:39 PM PST
Match ServerName: AdminServer
Match RuleType: Harvester
Match Rule: <${ServerRuntime//[weblogic.management.runtime.WebAppComponentRuntimeMBeanIcon.bea:ApplicationRuntime=browsestore,Name=AdminServer_/browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime//OpenSessionsCurrentCount]} >= 3)
Match Name: BrowsestoreWatch
Match DomainName: DiagnosticLab
Match AlarmType: None
Match AlarmResetPeriod: 60000
=====
Notification name: myjmx called. Count= 2.
Match severity: Notice
Match time: Feb 11, 2009 9:17:39 PM PST
Match ServerName: AdminServer
Match RuleType: Harvester
Match Rule: <${ServerRuntime//[weblogic.management.runtime.WebAppComponentRuntimeMBeanIcon.bea:ApplicationRuntime=browsestore,Name=AdminServer_/browsestore,ServerRuntime=AdminServer,Type=WebAppComponentRuntime//OpenSessionsCurrentCount]} >= 3)
Match Name: BrowsestoreWatch
Match DomainName: DiagnosticLab
Match AlarmType: None
Match AlarmResetPeriod: 60000
=====
```