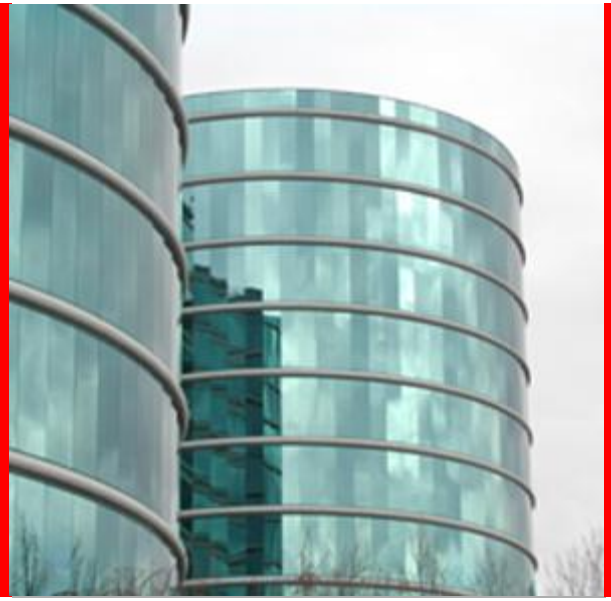
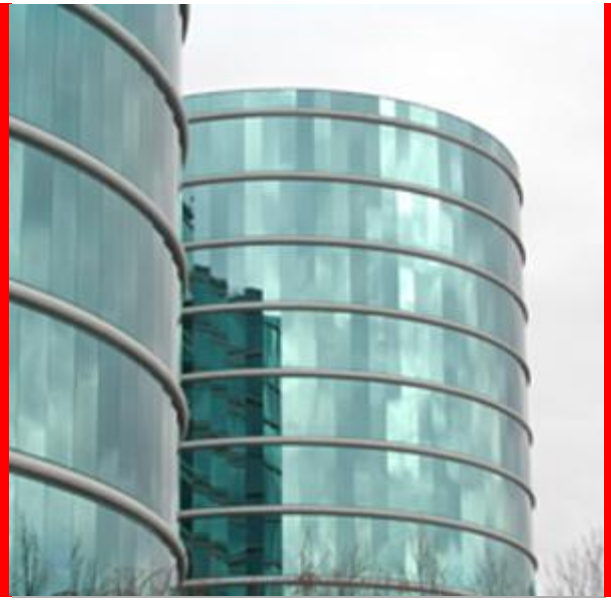


ORACLE®

WebLogic Diagnostic Framework

System and Application Diagnostics for Oracle WebLogic Server 11gR1



ORACLE®

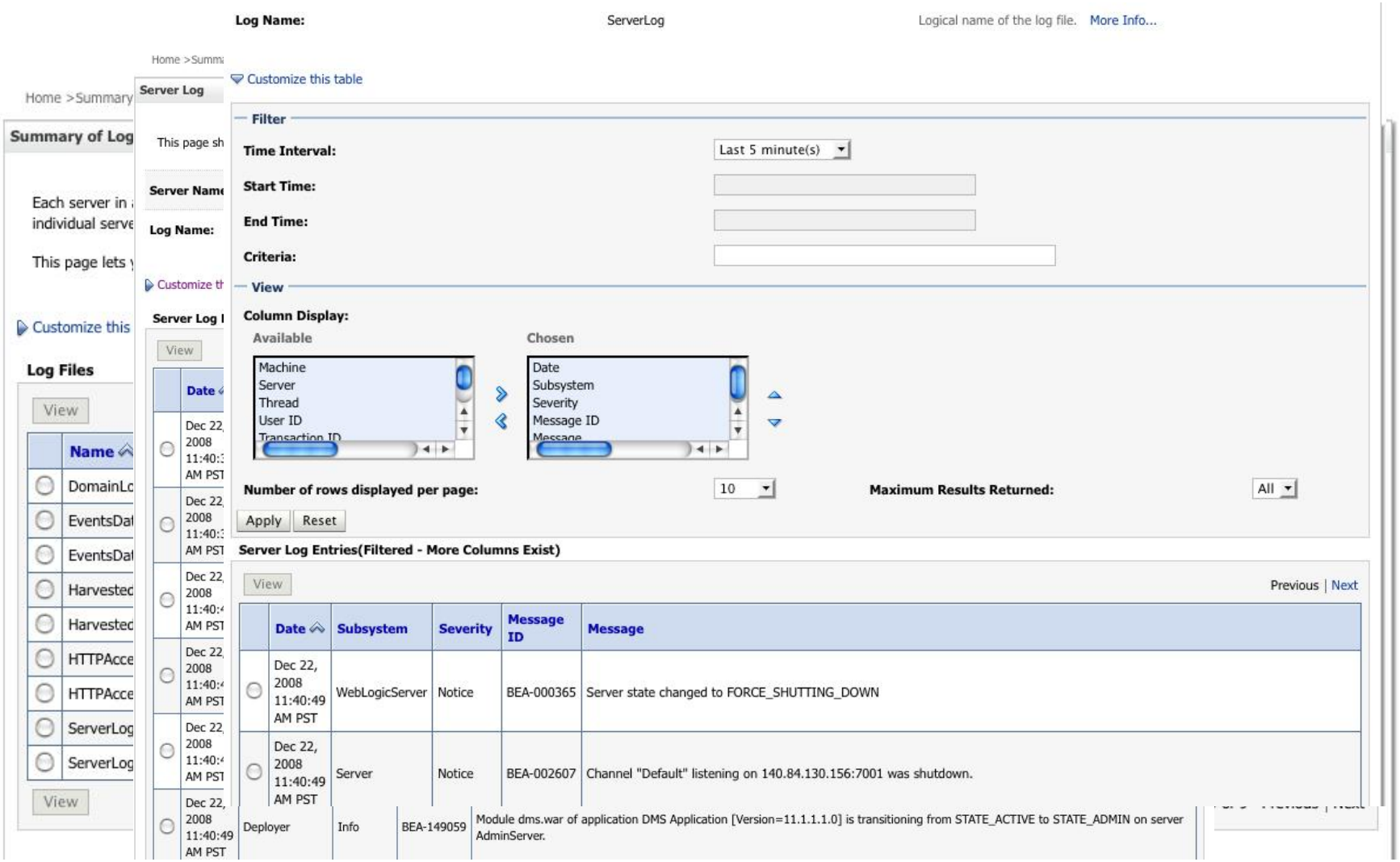
WebLogic Diagnostic Framework

Introduction and “How-to” Guide

Question: How do I control what I see in the WebLogic Server log files?

- Answer: By setting **Logging Levels and Filters**
- Summary: Use the admin console or WLST/JMX to configure what level of messages WLS will write to log files or stdout/stderr. You can also set filters to view only a subset of messages, e.g. those generated by a particular subsystem.
- Details: “Loggers and Handlers”

Example: WebLogic Log Files



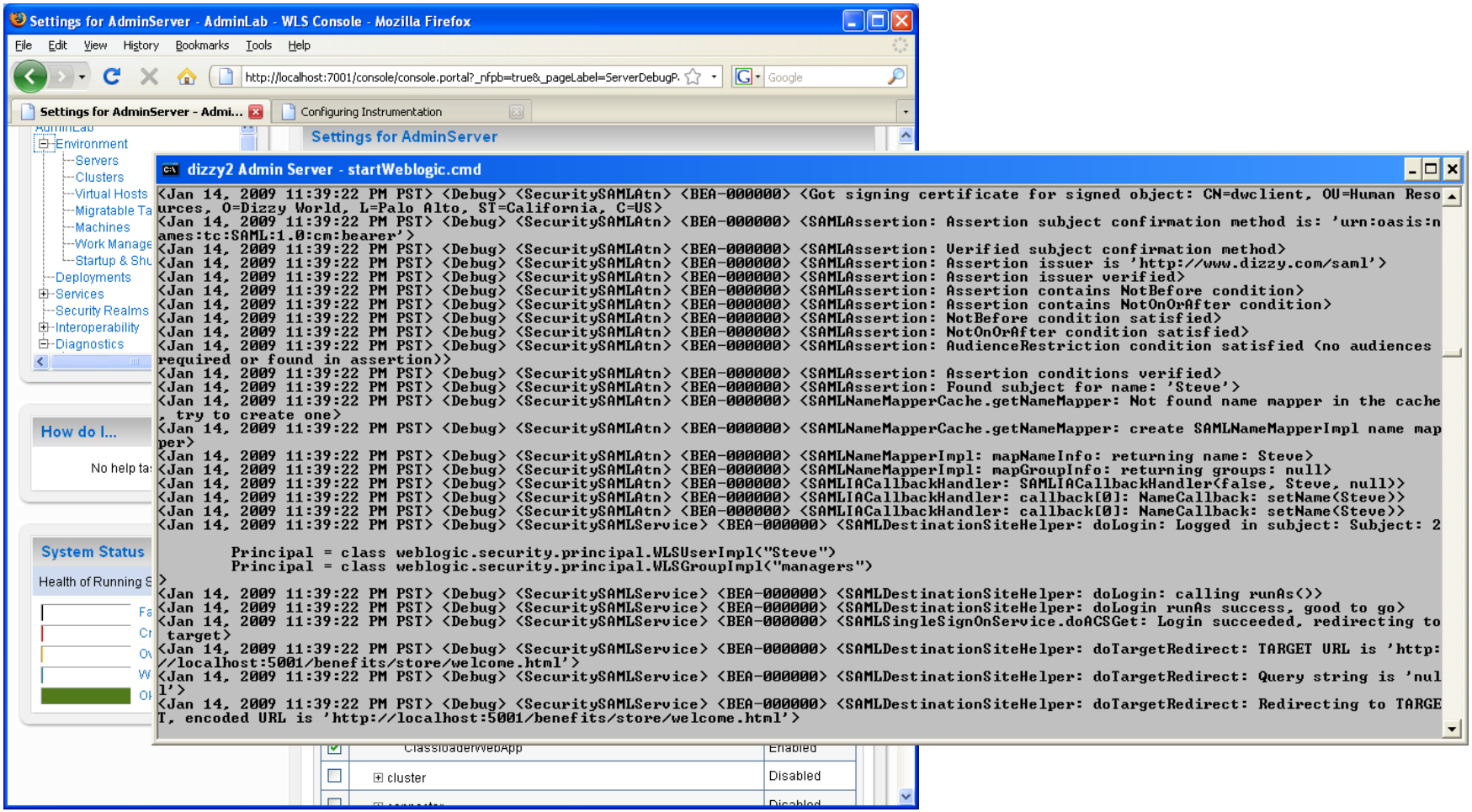
Question: How do I see Debug messages from WebLogic Server?

- Answer: Set Debug flags using the Admin Console.

- Summary: The WebLogic Server admin console allows you to enable/disable debug message logging for all the WLS runtime subsystems. These changes can be made dynamically and you can enable debugging at different levels of granularity (e.g. “jdbc”, “jdbc->sql”, “jdbc->rmi”, “jdbc->connection”) to specialize the messages.

- Lab: “Changing WebLogic Server Debug settings”

Example: Setting Debug Flags



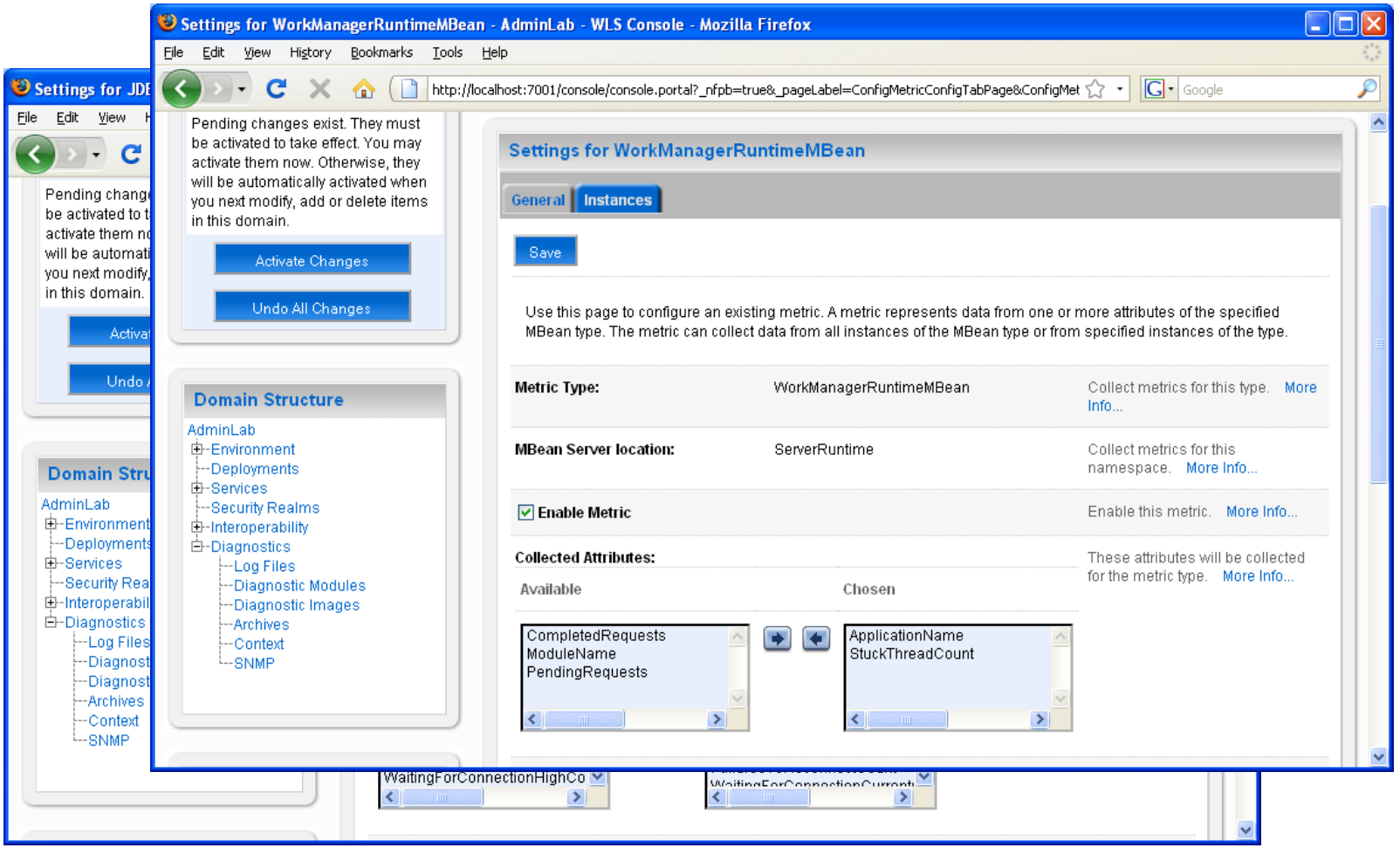
Question: How can I get a custom view of the WebLogic Server runtime?

- Answer: By configuring **Data Harvesters** to collect event data and metrics using WLS MBeans

- Summary: You can configure WLDF to harvest data from WebLogic Server's runtime MBeans. This data can then be written to log files or sent via notification messages in various ways. Information is available about all the WebLogic Server runtime subsystems.

- Details: "Accessing Diagnostic Data"

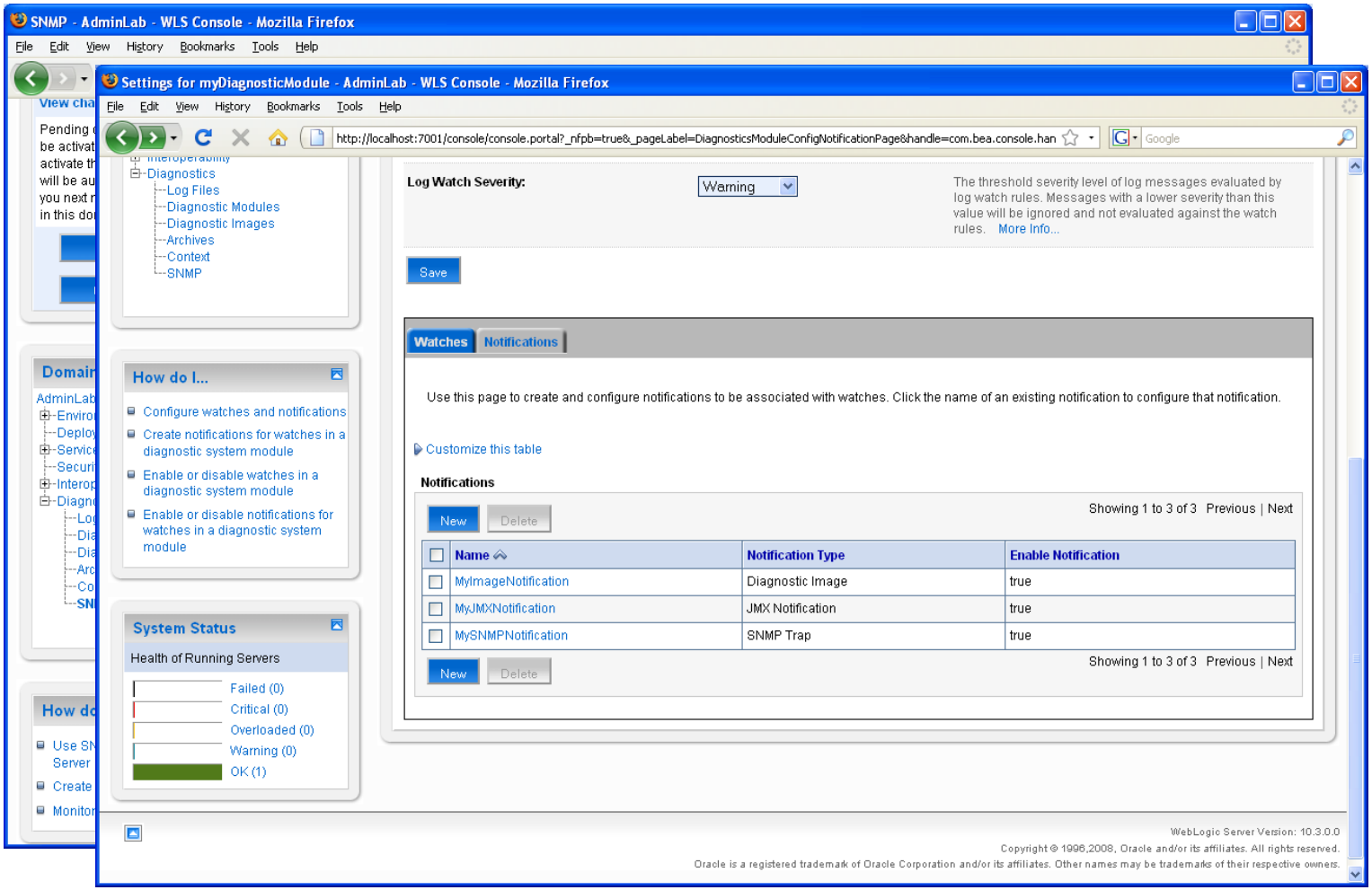
Example: Harvested Metrics



Question: Can I get a notification message when an event or condition occurs?

- Answer: **WLDF Watches and Notifications**
- Summary: You can set Watch rules that will cause WLDF to issue notification message via SNMP, SMTP, JMX, JMS in response to a set of events or conditions.
- Details: “Watches and Notifications”

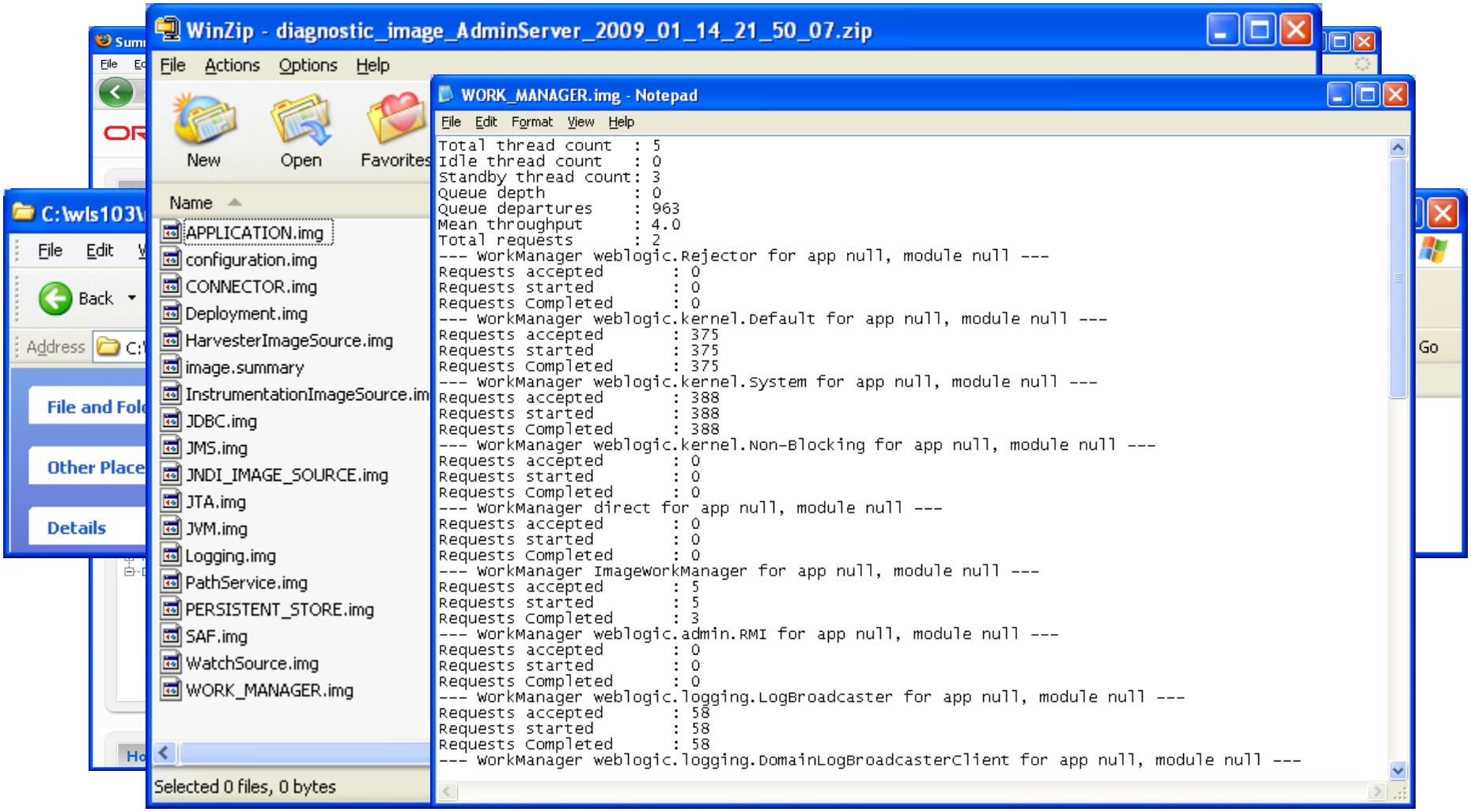
Example: WLDF Notifications



Question: How do I get a complete snapshot of the system state for support?

- Answer: WLDF **Diagnostic Image Capture**
- Summary: You can configure WLS to take a complete diagnostic snapshot of server runtime on demand (via the admin console or via WLST/JMX) or in response to a particular event or condition
- Details: “Diagnostic Image Capture”

Example: Diagnostic Image Capture



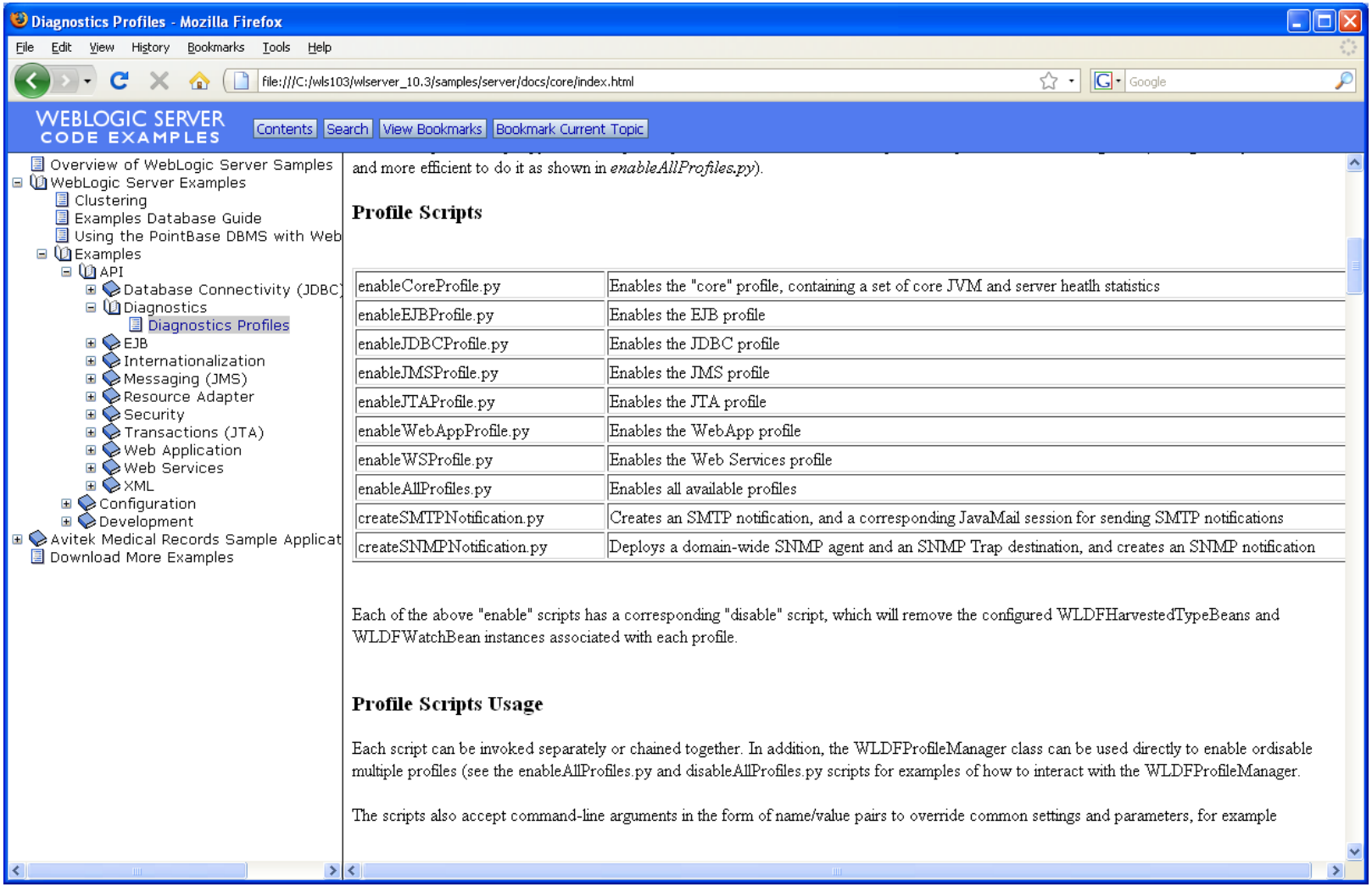
Question: Are there pre-built diagnostic monitors that I can simply re-use?

- Answer: WebLogic Server ships with a complete set of example **WLDF diagnostic profiles**

- Summary: WLS provides example WLST scripts that can be used to gather key subsets of diagnostic data about the major runtime subsystems (EJB, JDBC, JMS, JTA, Web App and WS) or any combination. There are also scripts for creating SMTP/SNMP notifications. You can use these “as is”, or customize them for your needs

- Details: WLS examples -> “diagnostics”

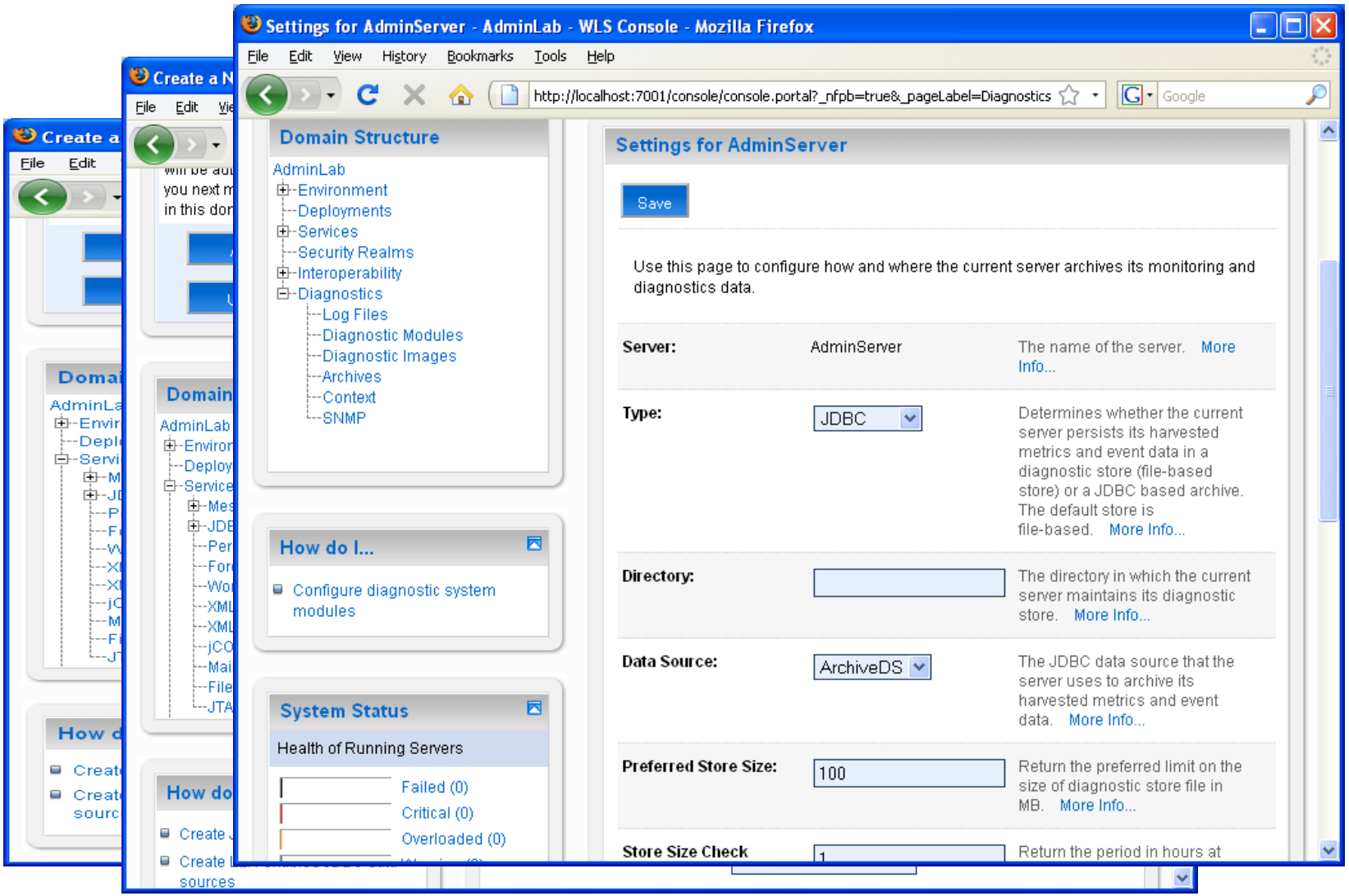
Example: Diagnostic Profiles



Question: How do I capture diagnostic data in an Oracle database and run queries?

- Answer: Configure a **JDBC diagnostic archive**
- Summary: You can configure the diagnostic archive to use a JDBC data store to store information in an external database. You can then run SQL queries to extract and re-present the data.
- Details: “WLDF Diagnostic Archives”

Example: JDBC Diagnostic Archive



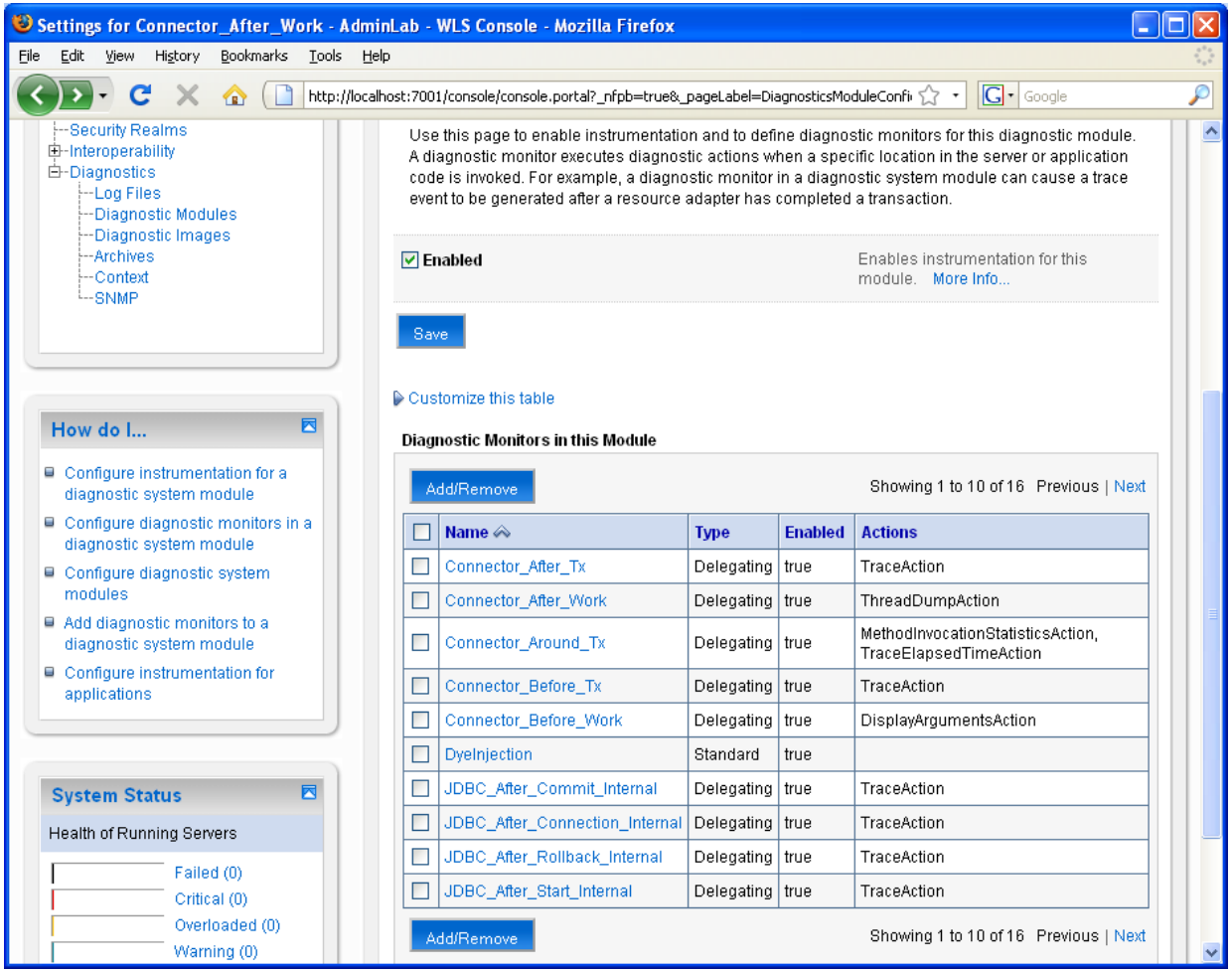
Question: How do I trace my application's execution inside WLS?

- Answer: **Diagnostic Actions** and **Diagnostic Monitors**

- Summary: You can configure Diagnostic Monitors, which will cause WLDF to generate trace messages. There are Standard, Delegating and Custom monitors, which allow you increasing levels of control over when and what the traces include: for example, log event, log elapsed time, dump method arguments, thread dump, stack trace.

- Details: "Diagnostic Monitors"

Example: Diagnostic Monitors



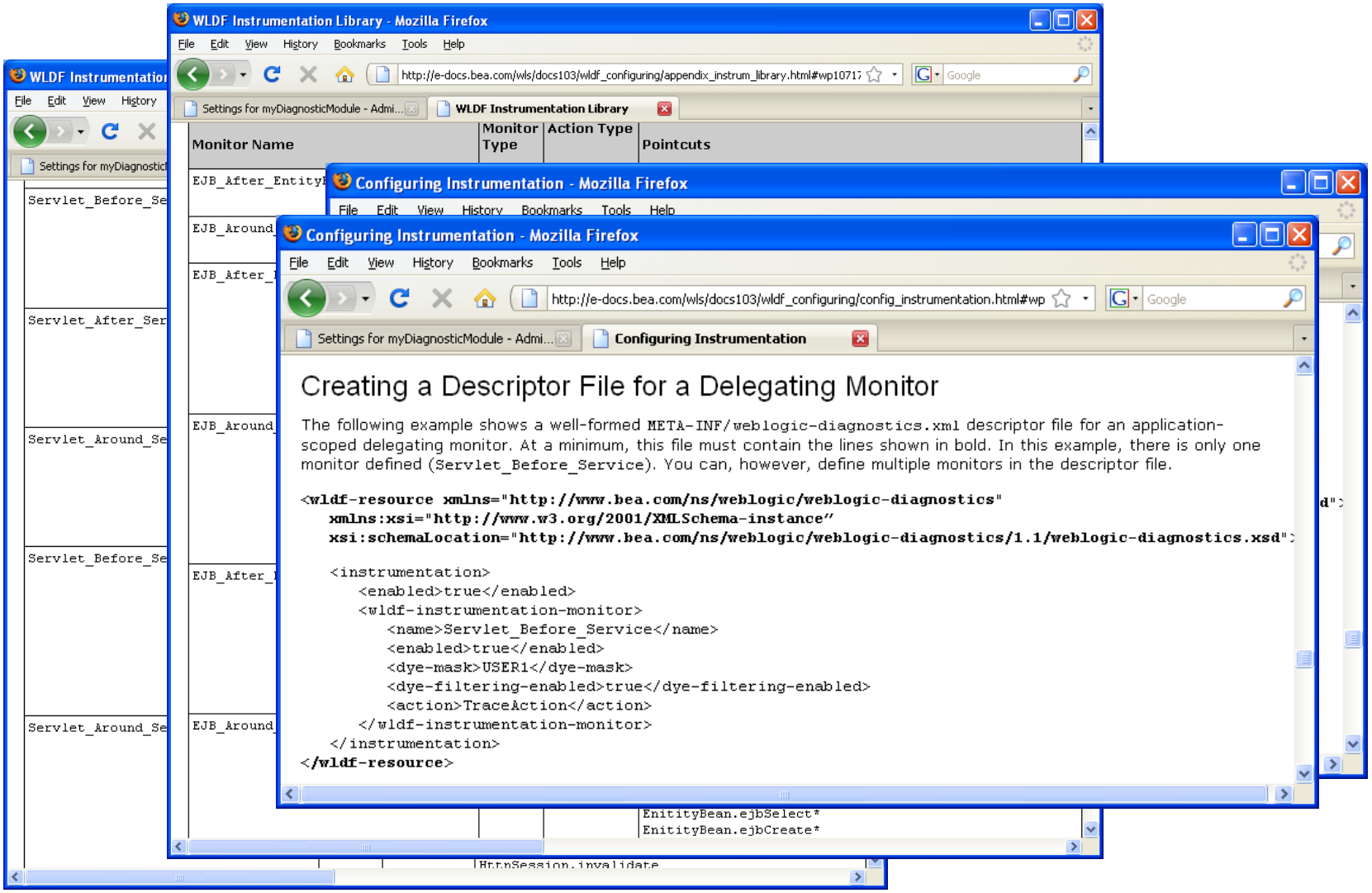
Question: Can I instrument my code so that I can trace its execution in the WLS log files?

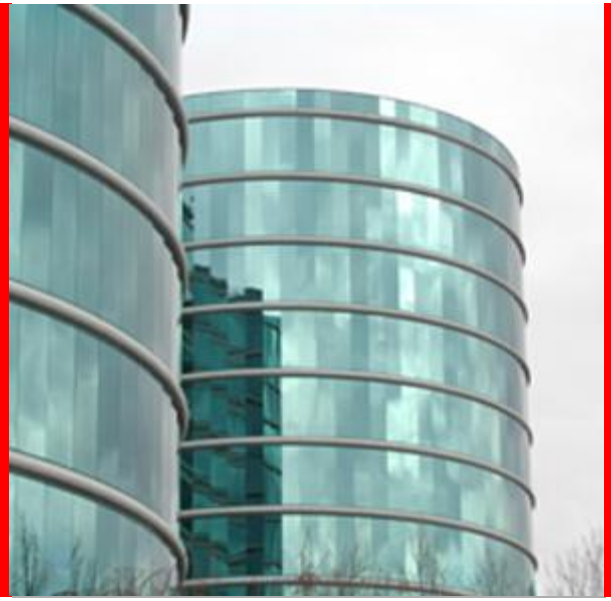
- Answer: **Application-scoped Instrumentation**

- Summary: You can define WLDF Diagnostic Monitors that allow you to trace execution of your code through messages in the WLS log files. WLDF uses Aspect-oriented programming (AOP) techniques to allow you to specify when and what messages are logged. You don't have to change your code to use this facility.

- See: "Application-scoped Instrumentation"

Example: App-Scoped Instrumentation





ORACLE®

WebLogic Diagnostic Framework

Overview and Concepts

WebLogic Diagnostic Framework

- Provides a co-coordinated, structured collection of tools that provide diagnostic and monitoring facilities
- Services run within the WebLogic Server process and participate in the server lifecycle
- Allows you to create, collect and analyze data on the runtime performance of servers and applications
- Enables dynamic access to runtime data through standard interfaces

WLDF Services

- **Data Creators** – generate diagnostic data
- **Data Collectors** – control persistence of diagnostic data and automated monitoring
- **Logger Service** – controls event logging
- **Harvester Service** – gathers harvestable metrics
- **Accessor Service** – exposes current and historic diagnostic data
- **Manager Service** – configuration and control
- **Image Capture Service** – provides diagnostic snapshot of server state (on-demand or triggered)

Diagnostic Data

- Event Data - synchronous event data generated by data publishers and collected by the Logger service
  - Application instrumentation (publisher)
  - Server code base logging
- Metric Data – data creators are sampled at regular intervals by the Harvester service to harvest current values
  - Application instrumentation (provider)
  - WLS runtime MBeans
  - Custom MBeans

Accessing Diagnostic Data

- The Accessor Service provides access to all WLDF event and metric data
- Online access
  - JMX-based access service to running server
  - Current state of harvestable values via Harvester services
  - Logged event data and persisted metrics via Archive service
- Offline access
  - Historical event and metric data via Archive service
- Time-based filtering by type, component, attribute
- Event filtering by severity, source and content

Automated Monitoring

- Watch Rules monitor event data from a data publisher (e.g. a logging filter) or metric data from a data provider collected by the Harvester service (e.g. an MBean)
- Watches can consist of multiple Watch Rules
- Watches can have one or more Notifications
  - Event logged in server log by default
  - SMTP, SNMP, JMX, JMS notifications supported

Configuring WLDF

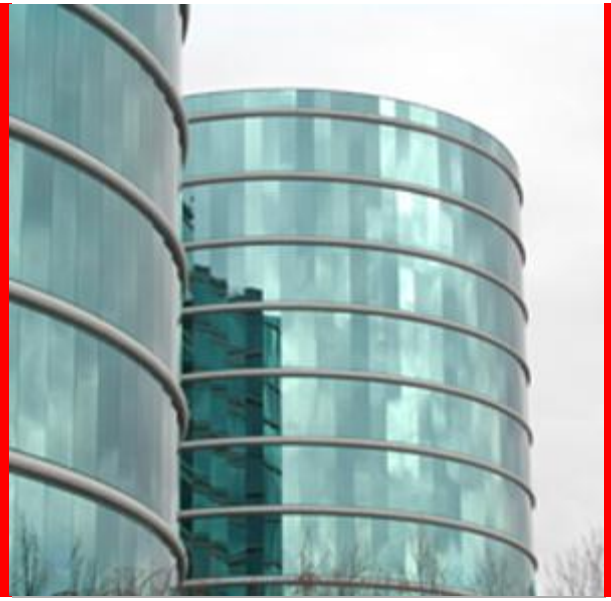
- WLDF is configured using edit and domain MBeans and the config is persisted in XML configuration files
- Configured using:
  - Admin console
  - WLST scripting
  - Programmatically via JMX and WLDF MBeans
  - Editing XML configuration files

Configuring WLDF Components

WLDF Component	Configured and Persisted
Image Capture Archive	Configured per server instance in domain *config.xml*
Harvester Watch and Notification Instrumentation (server)	Configured as a diagnostic module or resource deployed to a server instance. Persisted in diagnostic resource descriptor files
Instrumentation (application)	Configured in resource deployment descriptor deployed with the application. Persisted in application archive (e.g. .ear) file

Diagnostic System Module

- System resource used to configure Instrumentation, Harvester, Watch and Notification
- Only one Diagnostic System Module can be assigned to a WebLogic Server instance at a time
- Can be targeted to multiple servers within a domain
- Created as a WLDFResourceBean
- Persisted in resource descriptor file
  - <*domain*>/config/diagnostics/<*diag-module-name*>.xml
  - Schema from diagnostics.xsd
  - <wldf-system-resource> element in config.xml



ORACLE®

WebLogic Diagnostic Framework

Data Creators

Data Creators

- Data Publishers synchronously generate events (PUSH)
- Data Providers are sampled at regular intervals to harvest current values (PULL)

- MBeans:
  - encapsulate the configuration or current state of a managed resource
  - Can be accessed/manipulated programmatically or via admin tools

- Instrumentation:
  - adds code to WLS instances and applications to execute diagnostic actions at specified locations
  - tracks diagnostic context by identifying requests and tracking them as they flow through the system

WebLogic JMX and MBeans

- The Java Management Extensions (JMX) specification defines an architectural framework for managing JEE applications
- Access to devices (containers, subsystems…) is via MBeans
- Each WLS instance has an MBean Server containing domain, edit and runtime MBeans for that server
- Each managed resource (MBean) exposes attributes or runtime operations that encapsulate the configuration and current state of that resource

Configuration MBeans

- Reside only on the Admin Server
- Domain Config MBeans
  - Clusters, servers, containers
- System Module MBeans
  - Modules available to all applications on a server
  - Application-scoped modules not configured via JMX
- Security MBeans
  - WebLogic security providers
  - WebLogic security utilities (e.g. user account management)

Runtime MBeans

- Provide run-time information about a managed resource
- Exist only on the Managed Server to which the resource is deployed
- Are transient: runtime state data not persisted
- Accessed via Admin Server

MBean Servers

- MBean Server = container for MBeans
- WebLogic Server JVMs contain:
  - Domain MBean Server (Admin Server only)
  - Edit MBean Server (Admin Server only)
  - Runtime MBean Server (Admin/Managed Servers)
  - [optional] Platform MBean Server (WLS Runtime MBean Server can be configured as Platform MBean Server)

- WebLogic Server provides MBeans for managing all aspects of the application server and its services (e.g. JDBC, JTS, JMS, SNMP, XML …)
- http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e13951/core/index.html

Browsing MBeans

- WLNav (WebLogic Navigation Tool)
  - Open source project: formerly dev2dev.bea.com
  - Download from: http://wlnav.open.collab.net/
  - Deploy with: `java weblogic.Deployer -username <weblogic> -password <weblogic> -targets <AdminServer> -deploy wlnav.war`
  - To run: http://localhost:7001/wlnav
- WLST (WebLogic Scripting Tool)
  - Integral part of WLS – included in install
  - To run: `java weblogic.WLST`
  - Documentation: http://download.oracle.com/docs/cd/E17904_01/web.1111/e13813/reference.htm

WLDF Instrumentation Concepts

- Joinpoint:
  - a well-defined point in program flow where diagnostic code can be added
- Pointcut:
  - a well-defined set of joinpoints
  - WLDF Instrumentation provides a mechanism for allow the execution of specific diagnostic code at pointcuts
- Diagnostic Location:
  - A postion relative to a joinpoint where the diagnostic activity will take place
  - Examples: *before*, *after* and *around*
- Diagnostic Actions:
  - Specify the code to execute at a joinpoint

Diagnostic Actions

- *DisplayArgumentAction*
  - Captures method argument/return value to Event Archive
- *StackDumpAction*
  - Captures stack trace as payload to Event Archive
- *ThreadDumpAction*
  - Only with JRockit JVM (1.5+). Captures thread dump as payload to Event Archive
- *TraceAction*
  - Generates a trace event at the program location
- *TraceElapsedTimeAction*
  - Captures timestamps before and after execution of an associated joinpoint

Diagnostic Monitors

- Define where diagnostic code will be added and what diagnostic actions will be executed
- Take effect at server startup for
  - server classes
  - application classes (also on redeployment)
- WLS provides a library of diagnostic monitors

Type	Scope	Pointcut	Location	Action
Standard	Server	Fixed	Fixed	Fixed
Delegating	Server or Application	Fixed	Fixed	Configurable
Custom	Application	Configurable	Configurable	Configurable

Delegating Diagnostic Monitors

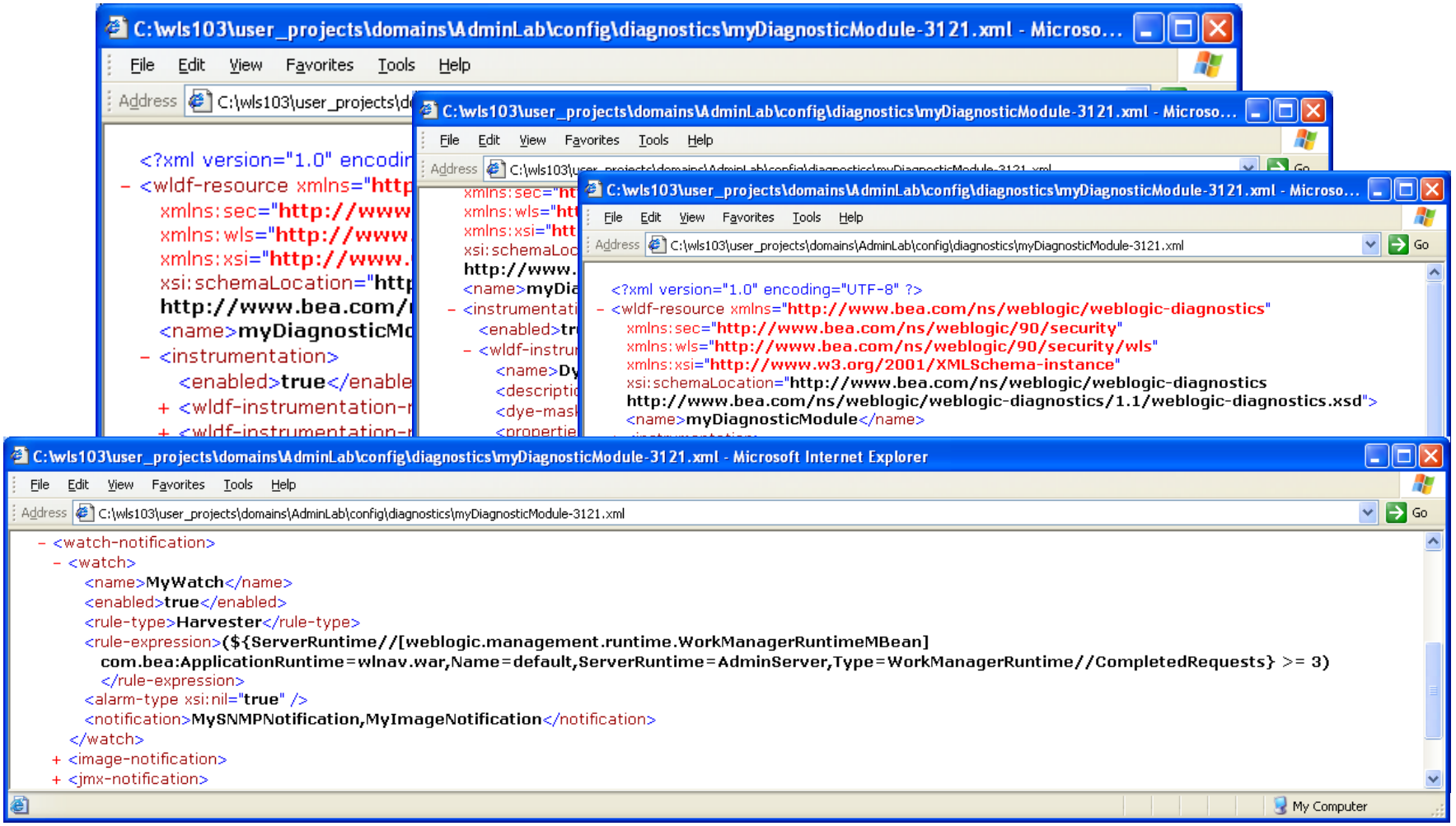
- Pointcut and location fixed, but can be configured to use any of the predefined diagnostic actions
- Some examples:

Monitor	Type	Pointcuts
Connector_Before_Inbound	Before	At entry of methods handling inbound connections
Connector_Around_Tx	Around	At entry and exit of transaction register, unregister, start, rollback and commit methods
JDBC_Before_Start_Internal JDBC_After_Rollback_Internal JDBC_After_Commit_internal	Before After After	Before and after JDBC transactions

Diagnostic Descriptor Files

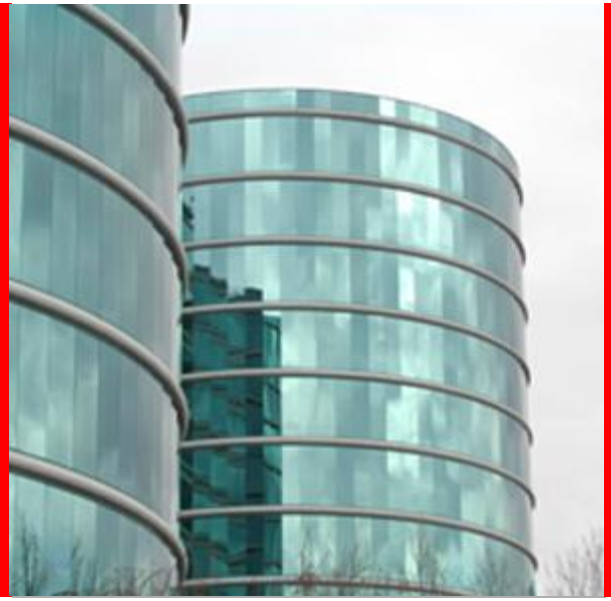
- XML diagnostic descriptor files contain configuration for:
  - Instrumentation
  - Harvester
  - Watch and Notifications
- Kept in <*domain_name*>*/config/diagnostics* directory
- Only one diagnostic descriptor file active at a time; cross-referenced from config.xml
- Diagnostic monitors configured via <*instrumentation*> stanza in diagnostic descriptor

Example: Server-Scoped Diagnostic Module



Diagnostic Context

- Contextual information added to requests to allow you to reconstruct transactional events – “event dyeing”
- Reconstruct a thread of execution request->response
- Keeps diagnostic volumes/overhead manageable
- Requests can be dyed for conditional tracing
- Examples dye flags:
  - ADDR1-4: used to dye requests per IP address
  - COOKIE1-4: used to dye requests using cookie values
  - USER1-4: used to dye requests from named users
  - PROTOCOL_xxx: used to dye requests using given protocol
  - DYE_0-7 reserved for application use:
    - http://download.oracle.com/docs/cd/E17904_01/web.1111/e13714/config_context.htm#WLDFC270



ORACLE®

WebLogic Diagnostic Framework

WLDF Logging and Harvesting

WLDF Logging

- WebLogic logging service provides all logging events to WLDF in a common format
- Can configure WLDF to collect/analyze/archive all events generated by WebLogic logging service
- WLS subsystems use WebLogic catalog APIs or Commons Logging APIs to generate messages
- Log handlers:
  - Stdout handler – writes to stdout
  - File handler – writes to log file(s)
  - Domain Log Broadcaster – logs to Admin Server domain log
  - JMX Log Broadcaster – broadcasts to JMX listeners
  - Memory Handler – admin console and server image capture

Message Formats

- Prefix (####) *[log messages only]*
- Locale-formatted timestamp
- Severity
- Subsystem
- Machine Name *[log messages only]*
- Server Name *[log messages only]*
- Thread ID *[log messages only]*
- User ID *[log messages only]*
- Transaction ID *[log messages only]*
- Diagnostic Context ID *[log messages only]*
- Raw Time Value *[log messages only]*
- Message ID
- Message Text

Message Severity Levels

- TRACE = Diagnostic Action library message
- INFO = Low-level informational message
- NOTICE = Higher-level informational message
- WARNING = Suspicious condition message
- ERROR = User error message
- CRITICAL = System or service error message
- ALERT = Service unusable; no automatic recovery
- EMERGENCY = Server unusable
- DEBUG = WebLogic logging service message

Controlling WLS Logging

- Can interact with LogMBeans via the admin console or WLST
- Can be specified on the command line:
  - -Dweblogic.log.StdoutSeverity=Debug
  - -Dweblogic.log.RedirectStdoutToLogEnabled=true

- Log Rotation
  - Default = 500KB (dev), 5000KB (prod)
  - Can change log rotation settings via admin console or WLST
  - Can configure log files to include date/time stamp
  - Force immediate rotation using LogRuntime.forceLogRotation()
  - Rotate based on size/time
  - Configure maximum number of log files

Filtering WLS Logging Messages

- Filtering options for messages to:
  - WLS log files
  - WLS stdout
  - Client JVM log files
  - Client JVM stdout
- Log filters configured via admin console, WLST or LogFilterMBean
- Filter on: RECORDID, DATE, SEVERITY, SUBSYSTEM, MACHINE, SERVER, THREAD, USERID, TXID, CONTEXTID, TIMESTAMP, MSGID, and MESSAGE using boolean/relational operators
- Creating log filters:
  - http://download.oracle.com/docs/cd/E15523_01/apirefs.1111/e13952/taskhelp/domain_log_filters/CreateLogFilters.html

WLDF Harvester

- Used to collect, analyze and archive the metrics contained in server and custom MBeans
- To configure a WLDF harvester:
  - Create a Diagnostics Module via the admin console:
  - Domain->Diagnostics->Diagnostic Modules
  - Collected Metrics->New
  - Select Runtime or Custom MBeans to harvest
  - Configure attributes/expressions for harvesting
- XML configuration file written to <*domain*>/config/diagnostics

Watches and Notifications

- Notification = action taken when a Watch rule expression evaluates to true
  - JMX, JMS, SNMP, SMTP, Diagnostic Image Capture
- Configured using admin console or WLST
  - Entries in system resource configuration file
    - <watch-notification>
    - <log-watch-severity>
    - <watch>
    - <jms-notification>
    - <jmx-notification>
    - <smtp-notification>
    - <image-notification>

JMX Notifications – How to

- Configure a JMX notification
- Configure a watch and assign to JMX notification
- Register a notification listener with the server's *WLDFWatchJMXNotificationRuntime* MBean to receive notifications

```
public void register() throws Exception {
  watchBroadcasterObjectName =
    new ObjectName(
      "com.bea:ServerRuntime=" + this.serverName + ",Name=" +
      JMXWatchNotification.GLOBAL_JMX_NOTIFICATION_PRODUCER_NAME +
      ",Type=WLDFWatchJMXNotificationRuntime," +
      "WLDFWatchNotificationRuntime=WatchNotification," +
      "WLDFRuntime=WLDFRuntime"
      );
  rmbs = getRuntimeMBeanServerConnection();
  addNotificationHandler();
}
```

SNMP Notifications – How to

- Configure server SNMP Agent and Trap Destination
  - Admin console: Domain->Diagnostics->SNMP
  - Trap destination: e.g. public, localhost, 161
- Create WLDF watch
- Create SNMP Notification and assign WLDF watch
- To receive traps, open shell window, set environment and run e.g.
  - java snmptrapd –d –p 161 –c public

SMTP Notifications – How to

- Configure a Mail Session
  - Admin console: Services->Mail Sessions
  - Configure JavaMail properties, e.g.:
    - mail.smtp.user=<user>
    - mail.host=<SMTP Server Address>
    - mail.transport.protocol=smtp
- Create a WLDF watch
- Create a SMTP notification and assign WLDF watch

JMS Notifications – How to

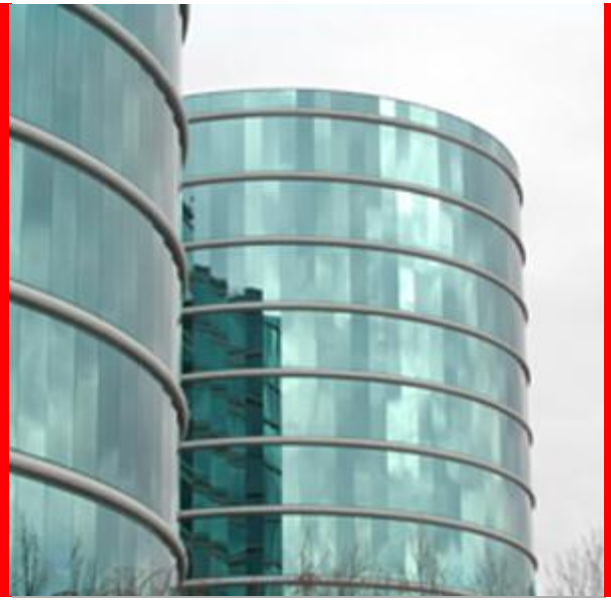
- Create JMS Server
  - Admin console: Services->Messaging->JMS Servers
- Create JMS Module
  - Admin console: Services->Messaging->JMS Modules
- Create JMS Queue + JNDI name
  - Admin console: Services->Messaging->JMS Modules->…
- Create a WLDF Watch
- Create a JMS Notification and assign the WLDF watch
  - Specify JMS Queue and JNDI name

Diagnostic Image Capture

- Created on-demand or automatically
- Can configure lockout period, to save system resources
- Captures:
  - Subsystem data (EJB, JMS, JDBC, JNDI, Web…)
  - Server configuration
  - Log cache state
  - JVM state
  - Work manager state
  - JNDI state
  - Harvestable data
- Image file has unique name (based on timestamp)
- Capture includes server, JVM, OS details plus date/time

Diagnostic Image Capture

- Designed for post-failure analysis
- Contents are pre-configured
- Default diagnostic image per server
- Can configure image file location and lockout period
- If defaults changed, config stored in config.xml:
  - <server-diagnostic-config>
- Triggered by:
  - Server failure
  - Diagnostic image watch notification
  - Direct API call via JMX
  - On-demand (admin console or WLST)



ORACLE®

WebLogic Diagnostic Framework

Accessing Diagnostic Data

WLDF Accessor Service

- Online: access via admin console, WLST or other tools (e.g. SQL utilities for RDBMS store)
- Accessor service interacts with:
  - The Harvester service for current data
  - The Archive service for historic data
- Programmatic access:
  - *WLDFAccessRuntimeMBean*
  - *WLDFDataAccessRuntimeMBean*

Supported Data Store Types

- Discovered/Accessed via *WLDFAccessRuntimeMBean*
  - HTTP_LOG
  - HARVESTED_DATA_ARCHIVE
  - EVENTS_DATA_ARCHIVE
  - SERVER_LOG
  - DOMAIN_LOG
  - HTTP_ACCESS_LOG
  - WEBAPP_LOG
  - CONNECTOR_LOG
  - JMS_MESSAGE_LOG
  - CUSTOM_LOG

Accessing Diagnostic Data

- Via admin console:
  - Domain->Diagnostics->Logs
- Programmatically via MBeans:

```
private MBeanServerConnection rmbs = null
...
ObjectName objName =
    new ObjectName( "com.bea:ServerRuntime=" + serverName +
        ",Name=Accessor," + "Type=WLDFAccessRuntime," +
        "WLDFRuntime=WLDFRuntime");
rmbs.getAttribute( objName, "WLDFDataAccessRuntimes" );
```

- Via WLST:
  - exportDiagnosticDataFromServer(*[options]*)

Accessing diagnostics via WLST

- wls:/mydomain/serverRuntime>exportDiagnosticsDataFromServer (“logicalName=HTTPAccessLog”, exportFileName=“myExport.xml”)
- Valid options include:
  - beginTimeStamp: earliest timestamp
  - endTimeStamp: latest timestamp
  - exportFileName: defaults to export.xml
  - logicalName: logical name of log file accessed
    - HarvestedDataArchive
    - EventsDataArchive
    - ServerLog
    - DomainLog
    - HTTPAccessLog
    - WebAppLog
    - ConnectorLog
    - JMSMessageLog
  - Query = WLDF query filter condition

Parsing and Displaying WLDF Data

- Check out Phil Aston's article on using XML and XSLT to parse and display WLDF diagnostic data:
- http://www.oracle.com/technetwork/articles/entarch/mining-wldf-xslt-083813.html
- For details of Data Store logical and column names:
- http://download.oracle.com/docs/cd/E17904_01/web.1111/e13714/appendix_query.htm#WLDFC330

- For WLST and WLDF example code:
- http://download.oracle.com/docs/cd/E17904_01/web.1111/e13714/appendix_wlst_ex.htm

WLDF Diagnostic MBeans

- WLDFRuntimeMBean
  - Provides access to all WLDF MBeans
- WLDFAccessRuntimeMBean
  - Access diagnostic data stores for a server
- WLDFDataAccessRuntimeMBean
  - Access diagnostic data for a specific data log
- WLDFArchiveRuntimeMBean
  - Statistical information about all WLDF diagnostic data archives
- WLDFDbstoreRuntimeMBean
  - Statistical information about diagnostic data stored in RDBMS
- WLDFFileArchiveRuntimeMBean
  - Statistical information about diagnostic data stored in files
- WLDFWlstoreArchiveRuntimeMBean
  - Statistical information about diagnostic data stored in WLS Store

- WLS MBean Reference:
  - http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e13951/core/index.html

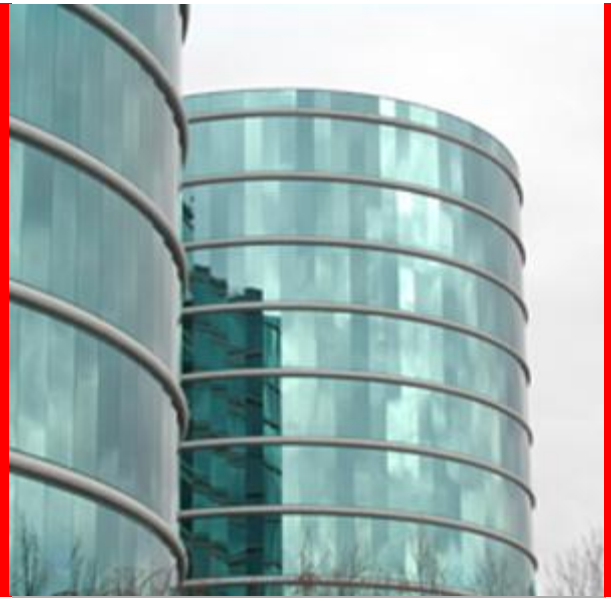
WLDF Diagnostic MBeans

- WLDFImageCreationTaskRuntimeMBean
  - Monitoring information about status of request for image capture
- WLDFImageRuntimeMBean
  - Controls diagnostic image capture
- WLDFHarvesterRuntimeMBean
  - Information about harvested and harvestable (current) diagnostic information, sampling and snapshot data
- WLDFInstrumentationRuntimeMBean
  - Runtime information about diagnostic instrumentation
- WLDFWatchNotificationRuntimeMBean
  - Provides access to watch and notification statistical data
- WLDFJMXNotificationListenerRuntimeMBean
  - Attach a JMX notification listener to receive WLDF notifications

- WLS MBean Reference:
  - http://download.oracle.com/docs/cd/E17904_01/apirefs.1111/e13951/core/index.html

WLDF Diagnostic Archives

- The Archiver service persists all data events, log records and metrics collected by WLDF
- Configured per-server in domain *config.xml*
- Can specify data retirement policy
- File-based archive (default)
  - Stored by default in: *<domain>/data/store/diagnostics*
  - *See <server-diagnostic-config>* in *config.xml*
  - Admin console: *Domain->Diagnostics->Archive*
- JDBC-based archive
  - Configure a data source via admin console as usual
  - Need tables wls_events and wls_hvst
  - DDL: http://download.oracle.com/docs/cd/E17904_01/web.1111/e13714/config_diag_archives.htm#i1067779



ORACLE®

WebLogic Diagnostic Framework

Application-Scoped Instrumentation

Server-Scoped Instrumentation

- Pre-defined joinpoints for gathering information
- Choice of standard or delegated monitors (pointcuts) for gathering information about:
  - Connectors
  - JDBC resources
- Delegated monitors are more flexible – you can specify which actions you want to take when the joinpoints are hit.
- Dye Injection allows you to target specific use cases

AspectJ: Joinpoints

- Similar to server-scoped instrumentation
- Uses AOP (aspect-oriented programming)
  - AspectJ is java implementation of AOP
  - Allows developer to insert functionality across all parts of an application without modifying actual code (“crosscutting”)
- Many joinpoints exist in AOP, but WLDF Instrumentation only supports the *method* joinpoint
- Joinpoint types: *call* vs. *execution*
  - Suggestion: use *execution* for tracing
  - See: http://www.eclipse.org/aspectj/doc/released/progguide/language-joinPoints.html for discussion of *call* vs. *execution*

AspectJ: Pointcuts

- Pointcut = A set of joinpoints and weaving rules
- Weaving = The process of combining business logic code and AOP code (weaving time = overhead to perform this)
- WLDF only supports *named* pointcuts (reusable)
- Pointcut syntax:

  [access-specifier] pointcut pointcut-name([args]) : pointcut-definition
- Pointcut Definition:
  - Specifies where to capture joinpoints
  - WLDF supports *method* and *constructor* pointcuts
  - Wildcards *, + and .. are supported
  - Example: public * CatalogBean.* (..) = all public methods in the CatalogBean class

AspectJ: Advice (WLDF Actions)

- AspectJ *advice* = actions to execute at joinpoint
- Referred to as *actions* in WLDF
- Actions available in WLDF (+ diagnostic location):
  - TraceAction (*before* and *after*)
  - DisplayArgumentAction (*before* and *after*)
  - TraceElapsedAction (*around*)
  - StackDumpAction (*before* and *after*)
  - ThreadDumpAction (*before* and *after*)

AspectJ: Aspects (WLDF Monitors)

- AspectJ *aspects* = like Java classes, with data, methods, nested class members, including pointcuts and advice
- WLDF uses **monitors** to specify pointcuts and advice
- WLDF uses **deployment descriptors** to define monitors
- DD *weblogic-diagnostics.xml*
  - Located *in meta-inf* directory of the application
  - Based on: http://www.oracle.com/technology/weblogic/90/diagnostics.xsd
  - <*instrumentation*> section only
- Application-scoped instrumentation can use delegating (= actions configurable) and custom (=poincut, location and actions configurable) monitors

Application-Scoped Delegating Monitors

- Many types of delegating monitors for applications, grouped by subsystem:
  http://download.oracle.com/docs/cd/E17904_01/web.1111/e13714/appendix_instrum_library.htm
- Custom monitors can be defined
  - in *weblogic-diagnostics.xml*
  - *<name>*
  - *<enabled>*
  - *<action>*
  - *<location>*
  - *<pointcut>*
- WebLogic Server creates the aspects using the AspectJ compiler (asj) and weaves the aspects in application source files when an application is deployed with a well-formed weblogic-diangostics.xml DD.

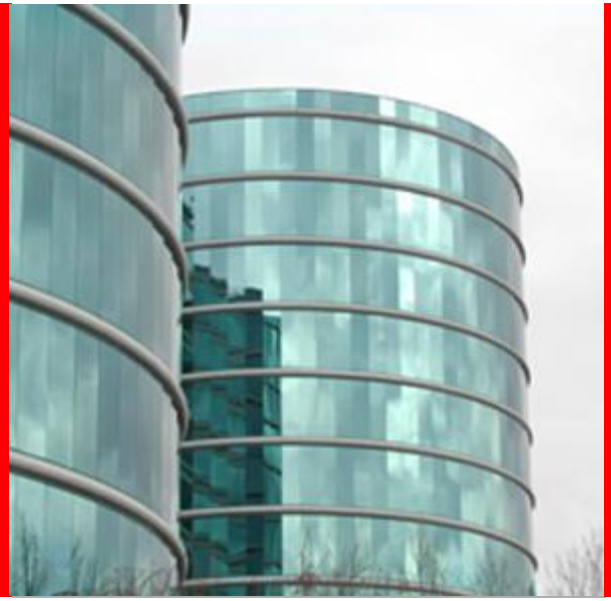
Instrumenting an Application: How-to

- Enable instrumentation on the server
- Create descriptor file *weblogic-diagnostics.xml*
  - Add delegating monitors
    - <wldf-instrumentation-monitor>
    - <enabled> set to true
    - <name> must match pre-defined values
    - <action> must match pre-efined actions
  - Add custom monitors (as above plus:)
    - <location> before, after or around
    - <pointcut> expression defining joinpoints in application
- Place well-formed descriptor file in META-INF for Java archive file (.ear, .war, .rar or .jar).

Updating Application Instrumentation

- WebLogic deployment plans + hotswap (Java 6) allow maximum flexibility for dynamic updates to application-scoped instrumentation
  - Add/detach monitors
  - Add/detach actions
  - Enable/disable monitors
- Create using weblogic.planGenerator

  http://download.oracle.com/docs/cd/E17904_01/web.1111/e13702/wlplangenerator.htm
- Can update via the admin console
  - Domain->Deployments->app->Confguration->Instrumentation



ORACLE®

WebLogic Diagnostic Framework

Loggers and Handlers

Logger and Handler Objects

- WLS subsystems generate status messages and pass them to java.util.logging.Logger objects
- WLS Logger objects publish WLLogRecord objects to message handlers subscribed to the Logger
- WLS instantiates a default set of handler objects
  - ConsoleHandler (admin console)
  - FileStreamHandler (log files)
  - Internal handler objects (domain log, JMX broadcast etc.)
- You can create and subscribe your own message handlers to the WLS Logger objects
  - Extend java.util.logging.Handler
  - http://download.oracle.com/docs/cd/E17904_01/web.1111/e13739/listening.htm#WLLOG185
- You can configure severity level and filter criteria for the both Logger and Handler objects

Setting Severity Levels for Loggers

- Use LoggingHelper API to get Logger class
  - getClientLogger (context = client JVM)
  - getServerLogger (context = serverJVM, server log)
  - getDomainLogger (context = admin server JVM, domain log)
- Set level using Logger.setLevel (Level level)
  - weblogic.logging.WLLevel maps to WLS levels
  - e.g. setLevel (WLLevel.ALERT)
  - http://download.oracle.com/docs/cd/E12840_01/wls/docs103/javadocs/weblogic/logging/WLLevel.html

Setting Severity Levels for Handlers

- Set through admin console or
  - Domain->Environment->Servers->*server*->Logging->Advanced
- Set through LogMBean (via WLST)
  - cd(“Servers/AdminServer/Log/AdminServer”)
  - cmo.setSeverityLevel(“Info”)
- Programmatically (via LoggingHelper API)
  - Logger.getHandlers() returns array of registered Handlers
  - Handler.getClass().getName() to get type
- Set level using Handler.setLevel (Level level)
  - weblogic.logging.WLLevel maps to WLS levels
  - e.g. setLevel (WLLevel.ALERT)

Setting Filters for Loggers/Handlers

- Set a Filter for Logger objects using Logging API
- Set a Filter for Handler objects
  - Admin Console
  - WLST (via MBeans)
  - Logging API – java.util.logging.Filter (see Example slide)

Example: Setting Console Handler Level via API

```
import java.util.logging.Logger;
import java.util.logging.Handler;
import weblogic.logging.LoggingHelper;
import weblogic.logging.WLLevel;

public class LogLevel {
 public static void main(String[] argv] throws Exception {
  Logger serverlogger = LoggingHelper.getServerLogger();
  Handler[] handlerArray = serverlogger.getHandlers();
  for (int i=0; i<handlerArray.length; i++) {
   Handler h = handlerArray[i];
   if (h.getClass().getName.equals("weblogic.logging.ConsoleHandler")) {
     h.setLevel(WLLevel.ALERT);
}}}}
```

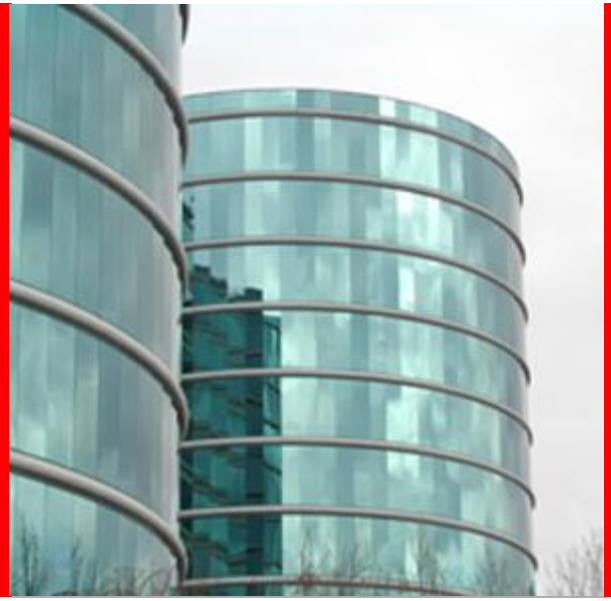
Example: Setting Stdout Handler Level via WLST

C:\>**java weblogic.WLST**
wls:/offline> **connect('weblogic','weblogic')**
wls:/AdminLab/serverConfig> **edit()**
wls:/AdminLab/edit> **startEdit()**
wls:/AdminLab/edit !>**cd("Servers/AdminServer/Log/AdminServer")**
wls:/AdminLab/edit/Servers/AdminServer/Log/AdminServer !>
cmo.setStdoutSeverity("Info")
wls:/AdminLab/edit/Servers/AdminServer/Log/AdminServer !>**save()**
wls:/AdminLab/edit/Servers/AdminServer/Log/AdminServer !>**activate()**

Example: Setting Filter Level via API

```
import java.util.logging.Logger;
import java.util.logging.Filter;
import java.util.logging.LogRecord;
import weblogic.logging.WLLogRecord;
import weblogic.logging.WLLevel;

public class MyFilter implements Filter {
  public boolean isLoggable (LogRecord record) {
    if (record instanceof WLLogRecord) {
      WLLogRecord rec = (WLLogRecord)record;
      if (rec.getLoggerName().equals("Deployer")) {
        return false;
      } else {
        return true;
      }
    } else { return false; }
  }
}
```



ORACLE®

WebLogic Diagnostic Framework

Sample Diagnostic Profiles

Example Diagnostic Profiles

- WebLogic Server ships with a set of pre-built ‘diagnostic profiles’
- Common WDLF configurations that cover key WebLogic Server subsystems
- Set of Jython (.py) classes encapsulating commonly-used Harvester and Watch metrics for each area, with utility classes for working with the WLDF framework
- Profiles: Core (JVM and server health), JDBC, JMS, EJB, JTA, WebApp, Web Services
- Notifications: SNMP, SMTP

<wls_install>/wlserver_10.3/samples/server/docs/core/index.html