ORACLE®

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Oracle Training Materials – Usage Agreement

Use of this Site (“Site”) or Materials constitutes agreement with the following terms and conditions:

1. Oracle Corporation (“Oracle”) is pleased to allow its business partner (“Partner”) to download and copy the information, documents, and the online training courses (collectively, “Materials") found on this Site. The use of the Materials is restricted to the non-commercial, internal training of the Partner’s employees only. The Materials may not be used for training, promotion, or sales to customers or other partners or third parties.

2. All the Materials are trademarks of Oracle and are proprietary information of Oracle. Partner or other third party at no time has any right to resell, redistribute or create derivative works from the Materials.

3. Oracle disclaims any warranties or representations as to the accuracy or completeness of any Materials. Materials are provided "as is" without warranty of any kind, either express or implied, including without limitation warranties of merchantability, fitness for a particular purpose, and non-infringement.

4. Under no circumstances shall Oracle or the Oracle Authorized Boot Camp Training Partner be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this Site of Materials. As a condition of use of the Materials, Partner agrees to indemnify Oracle from and against any and all actions, claims, losses, damages, liabilities and expenses (including reasonable attorneys' fees) arising out of Partner’s use of the Materials.

5. Reference materials including but not limited to those identified in the Boot Camp manifest can not be redistributed in any format without Oracle written consent.

ORACLE®

Oracle WebLogic Server 12c Implementation Boot Camp Manage Script Deploy

get resource
RESOURCES ARE COMPANY'S STRENGTH

ORACLE® PARTNERNETWORK
Specialized. Recognized by Oracle.
Preferred by Customers.

Administration Tools

- Configuration Wizard
  - GUI/scriptable tool to create and extend WebLogic domains
  - Template based
- Administration Console
  - Browser-based tool for configuring and monitoring domains, deploying applications, and controlling servers
- WebLogic Scripting Tool (WLST)
  - Script or command line tool to do the same thing as the Administration Console and Configuration Wizard
  - Note that we will cover details on WLST in a separate document
- weblogic.Admin
  - Deprecated command line tool for configuring a domain
  - Recommend using WLST instead
- weblogic.Deployer
  - Command line tool for deploying applications

Deployment

- Deployment units
  - JavaEE applications (Enterprise and web applications)
  - Shared Libraries and optional packages
  - Web services and EJBs
  - JavaEE modules (JMS, JDBC and WLDF etc)
    - Deployed as standalone resource or part of JavaEE app
  - Resource adapters, client app archives etc
- In what format
  - Archive
  - 'exploded directory' format

Deployment Options

- Two-phase deployment
  - Prepare phase – app distributed to all members of cluster and validated
  - Activate phase – deployed and active
    - In case of failure, starts in admin mode (only admin role can access it)
- Targeting
  - Apps are deployed via Administration Server which pushes the deployment settings out to all targeted managed servers
  - Target cluster rather than individual servers for easier management and to enable newly added servers to inherit all deployed apps automatically

Staged Deployment

- Determine how deployment files are made available to target servers
  - **Staged**
    - admin server pushes copy of app out to staging directory of managed servers (default)
    - Use for small and medium size apps to multiple servers/cluster
  - **No-Stage**
    - all servers reference same copy of app hosted on shared storage
    - Use for
      - Single server domain
      - Very large app on shared disk
      - Exploded archive which needs to be periodically redeployed
  - **External-Stage**
    - Administrator copies deployment files to staging directory, not WLS
    - Use where third party app/script manage software distribution

Deployment Mechanisms

- Administration Console
  - Complete control – configuration, deployment
- Command Line tools
  - WLST
    - Online deploy() function
    - WLST Offline Deploy e.g. create('MyApp', 'AppDeployment') function; useful for building domains with 'pre-deployed' apps
  - java weblogic.Deployer
  - Ant Task wldeploy
- IDE deploy to single server
  - JDEV, Eclipse, Netbeans
- Maven
- Auto deploy
  - for 'dev' mode, deploys to admin server only, no config in config.xml, deployment plan can not be used, can not be undeployed/redeployed using WLS tools

Production deployment

- Also called Side-By-Side Deployment
  - Two application versions can coexist
  - Test versions as administrator before opening up to users
  - Automatic/manual retirement: graceful, timeout (RMI), immediate

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -stop -adminmode -graceful
   -rmigraceperiod 30
```

  - By default, current user session is destroyed at the time of redeployment of web app.
    - Set save-sessions-enabled to true in container-descriptor of weblogic.xml to preserve user sessions

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -redeploy -name myApp
   -targets mymodule1@myserver1,mymodule2@myserver2
```

Production deployment

- Redeploying modules (requires exploded deployment)

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name mymodule -stop -adminmode -graceful
   -rmigraceperiod 30
```

- Redeploying static content (requires exploded deployment)

```
java weblogic.Deployer -adminurl http://localhost:7001 -user weblogic
   -password weblogic -name myApp -redeploy myApp/myjsps
```

- Application must satisfy some restrictions
  - Production redeployment only supports HTTP clients and RMI clients. Development must ensure that applications using production redeployment are not accessed by an unsupported client.
  - Enterprise applications can contain any of the supported Java EE module types. Enterprise applications can also include application-scoped JMS and JDBC modules.

Deployment Plan

- Allows customizable environment settings for parameters defined in deployment descriptor
- Stored outside of application archive
- Console automatically create plan when updating configuration values
- Command line tool to generate plan
- Plan can be specified in deployment time
  - e.g. Change the connection factories in adapter
- Limitations
  - Only parameters defined in deployment descriptor can be changed.
  - Not all items in DD can be customized in Deployment plan e.g. EJB name can not be changed

Deployment Plan

```
<?xml version='1.0' encoding='UTF-8'?>
<deployment-plan xmlns="http://www.bea.com/ns/weblogic/90"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/ns/weblogic/90
http://www.bea.com/ns/weblogic/90/weblogic-deployment-plan.xsd">
  <application-name>Temp</application-name>
  <variable-definition>
    <variable>
      <name>SessionDescriptor_timeoutSecs_12112996055620</name>
      <value>360</value>
    </variable>
  </variable-definition>
  <module-override>
    <module-name>jspExpressionWar</module-name>
    <module-type>war</module-type>
    <module-descriptor external="false">
    <root-element>weblogic-web-app</root-element>
    <uri>WEB-INF/weblogic.xml</uri>
    <variable-assignment>
    <name>SessionDescriptor_timeoutSecs_12112996055620</name>
    <xpath>/weblogic-web-app/session-descriptor/timeout-secs</xpath>
    </variable-assignment>
  </module-override>
     …..
</deployment-plan>
```

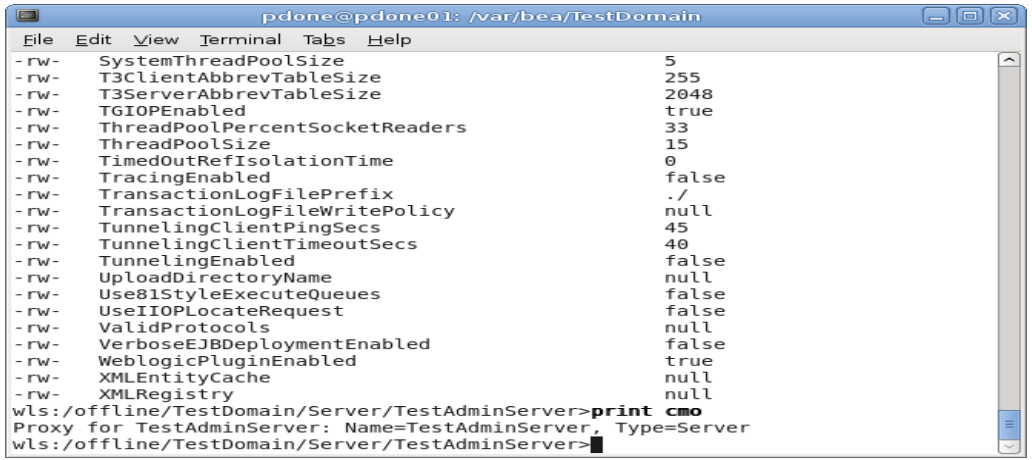
Scripting WebLogic & Deployments

Scripting

WebLogic Scripting Tool (WLST)

Command-line/scripting tools for WebLogic domains

- *Offline* (not connected to live server - primarily for domain creation)
- *Online* (connected to live server - primarily for runtime monitoring)
- *Interactive* mode (a command shell)



Jython based

- Python code in JVM
- WebLogic specific set of management APIs
- Easy access to WebLogic JMX API (in online mode only)

Scripting

The Value of WLST

Used by most customers

- Stable supported API for domain creation
- Easy access to WebLogic JMX MBeans
- Enables flexibility for generating dev/test/prod domain
- Beware: it has a learning curve (Jython, WLST APIs)

Repeatable automated process for domain creation

- Typically property file driven
- Greatly reduces opportunity for manual errors
- Single command to create a complex domain
- Pays-off even for scripting domains for the development lifecycle of project (not just for test/prod)
- Often integrated into 'continuous integration' processes

Two-Phase Configuration Changes

- Changes activated in batches:
  - Reliability, consistency:
    - Make (related) changes as a group
    - Validate before making the change
    - Activate or Roll back as a single unit (all changes on all servers)
- General process:
  - Get an edit lock
  - make changes
    - changes are stored in the pending directory
  - activate your changes (with implicit validation through the Admin Console or WLST)
    - changes are distributed to servers in the domain
    - Two phases: prepare and commit
    - Prepared on all servers; any failures will cause total rollback

Intro to WebLogic Scripting Tool (WLST)

WebLogic Scripting Tool (WLST)

- Scripting tool for administering a domain (create, configure, manage, monitor, deploy applications)
- Based on Jython – pure Java implementation of Python
- Great for automating repetitive tasks
- Heavily used by customers and within BEA

Interaction Modes

- Interactive
  - enter a command and view the response at a command-line prompt
  - In online mode: shell maintains a persistent connection to a WLS instance
- Script
  - text file with a .py file extension
  - executed using Jython commands for running scripts
  - invoke a sequence of WLST commands without requiring your input
- Embedded
  - instantiate the WLST interpreter in your Java code
  - execute WLST commands from a Java program

Connection Modes

- Offline: analogous to the Configuration Wizard
  - Uses the Offline Configuration Framework
    - Also used by the Configuration Wizard
    - Consistent results when using either tool
  - read and write access to the configuration data that is persisted in the domain's config directory or in a domain template JAR
  - Intended to create a domain or modify a non-running domain
  - Used during WLS install to create samples domains

- Online: analogous to the Administration Console
  - JMX client
  - Interacts with a server's MBeans
  - Intended as a runtime management tool: configuration, management, deployment, monitoring

WLST Offline

- Uses the Offline Configuration Framework
  - Also used by the Configuration Wizard; Consistent results when using either tool
- Uses domain templates to create a domain
  - Several shipped with WLS
  - Create your own using Template Builder
  - Modify an existing template using WLST Offline
- Intended to create a domain or modify a non-running domain
- Used during WLS install to create samples domains
- Used by WL Platform products for creating domain configurations (use specific templates for each product)

Note: Do not use WLST offline to manage the configuration of an active domain. Offline edits are ignored by running servers and can be overwritten by JMX clients such as WLST online or the WebLogic Server Administration Console.

WLST Offline – Two Models: Templates and Scripts

- Templates (put everything in the template)
  - Use Template Builder to capture current domain configuration and artifacts into a rich template
  - Use that template to create new domains, migrate from dev to production
  - Bundles required files
  - Used by WebLogic Platform products

- Scripts (put little in the template and most in scripts)
  - Use more basic templates for domain creation
  - Use WLST scripts along with some predefined or custom domain templates to create the target domain
  - Modify/customize domain configuration through scripts, and effectively track configuration changes through source control

Recommended for:
- **Out of the box config for a layered product**

Reason: domain is self-contained

Recommended for:
- **Customer configuration migration**
- **QA automation**

Reason: easily make and track changes to a domain config

Starting WLST

- Setup the environment:
  - setWLSEnv.sh/cmd – sets path and classpath
  - Adds WebLogic Server classes to the CLASSPATH and WL_HOME\server\bin to the PATH
- Invoke WLST:
  - java weblogic.WLST
  - or
  - java weblogic.WLST c:\myscripts\myscript.py
- Starts in Offline mode

Creating a Domain (WLST Offline)

- Syntax
  - createDomain(domainTemplate, domainDir, user, password)

- Example
  - wls:/offline> createDomain('c:/bea/wlserver_103/common/templates/domains/wls.jar','c:/mydomain', 'weblogic', 'weblogic')

Changing a Domain in WLST Offline

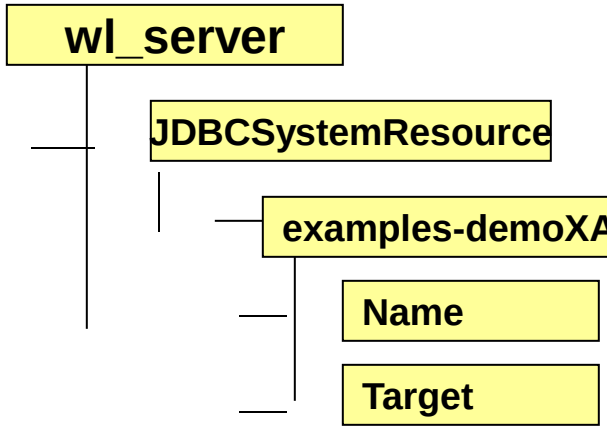
Step	Syntax
1. Open a domain for editing	`readDomain(`*`domainDirName`*`)`
2. Extend the domain (optional)	`addTemplate(`*`templateFileName`*`)`
3. Make changes (optional)	`Various commands`
4. Save	`updateDomain()`
5. Close the domain for editing	`closeDomain()`

Browsing and Editing in WLST Offline

- Browsing:
  - cd(), ls()
- Editing:
  - Add an application to a domain:
    - addTemplate(templateFileName)
  - Create and delete management objects:
    - create(name, childMBeanType)
    - delete(name, childMBeanType)
  - Get and set attribute values:
    - get(attrName)
    - set(attrName, value)
  - Set domain creation or update options:
    - setOption(optionName, value)
  - Load SQL files into a database:
    - loadDB(dbVersion, connectionPoolName)

WLST Offline – accessing a domain

- Offline – reading a domain:
  - readDomain('c:/bea/user_projects/domains/medrec')
- Domain configuration data is a collection of XML documents that expresses a hierarchy of management objects
- WLST represents this hierarchy as a file system
  - The root is the management object that represents the WebLogic Server domain (domain directory)
  - Each managed-object type is a sub directory of the root
  - Each instance of the type is a subdirectory under the type directory
  - Each management attribute and operation is a file within a directory



Syntax

- Command names and arguments are case sensitive.
- Enclose arguments in single or double quotes. For example, 'newServer' or "newServer".
- If you specify a backslash character (\) in a string, either precede the backslash with another backslash or precede the entire string with a lower-case r character.
  - For example when specifying a file pathname that contains a backslash:
    - readTemplate('c:\\userdomains\\mytemplates\\mytemplate.jar') or readTemplate(r'c:\userdomains\mytemplates\mytemplate.jar')
- When using WLST offline, the following characters are not valid in names of management objects: period (.), forward slash (/), or backward slash (\).
  - If you need to cd to a management object whose name includes a forward slash (/), surround the object name in parentheses. For example:
    - cd('JMSQueue/(jms/REGISTRATION_MDB_QUEUE)')

WLST Online

- Analogous to the Administration Console, but without the GUI
- JMX client; maintains a persistent connection
- Interacts with a server's/domain's MBeans
- Intended as a runtime management tool: configuration, management, deployment, monitoring

WLST Online – connecting to a domain

- Setup the environment:
  - setWLSEnv.sh (in WL_HOME\server\bin)
  - Adds WebLogic Server classes to the CLASSPATH and WL_HOME\server\bin to the PATH
- Invoke WLST:
  - java weblogic.WLST
- Starts in Offline mode
- Connect to a domain:
  - wls:/offline> connect('weblogic','weblogic','localhost:7001')

Traversing MBean Trees

- Simpler than JMX – no need to know the JMX object name
- MBeans are hierarchical, similar to a file system, with the DomainMBean at the top of the tree
- Multiple MBean trees (described later)
- Use commands similar to Unix commands to traverse the tree:
  - cd(), ls()
- Syntax is the same as with WLST Offline

- Domain MBean (root)
  - |- - - MBean type (ServerMBean)
    - |- - - MBean instance (ManagedServer1)
      - |- - - MBean attributes & operations (AutoRestart)
    - |- - - MBean instance (MedRecServer)
      - |- - - MBean attributes & operations (StartupMode)

Available MBean Trees

- domainConfig
  - configuration hierarchy of the entire domain; represents the configuration MBeans in RuntimeMBeanServer
  - read only
- serverConfig
  - configuration hierarchy (configuration MBeans) of the server your are connected to
  - read only
- domainRuntime
  - hierarchy of runtime MBeans for the entire domain
  - read only
- serverRuntime
  - hierarchy of runtime MBeans for the server you are connected to
  - read only
- edit
  - writable domain configuration with pending changes; represents the configuration MBeans in the EditMBeanServer
- jndi
  - read-only JNDI tree for the server you are connected to
- custom
  - list of custom MBeans
  - can be hierarchical/grouped if MBeans use namespaces appropriately

Switching Between Trees

- Use the appropriate command to move to a different tree
  - domainConfig()
  - serverConfig()
  - domainRuntime()
  - serverRuntime()
  - edit()
  - jndi()
  - custom()
- When returning to a tree, you return to the place where you left, except custom and jndi (goes to the root)

Changing Configuration in WLST Online

Step	Syntax
1. Change to the edit tree	`wls:/wl_server/domainConfig> edit()`
2. Get an edit lock	`wls:/wl_server/edit> startEdit()`

3. Make changes	`wls:/wl_server/edit !> svr = cmo.createServer("managedServer")`
	`wls:/wl_server/edit !> svr.setListenPort(8001)`
	`wls:/wl_server/edit !> svr.setListenAddress("my-address")`

4. Save (and implicitly validate) your changes	`wls:/wl_server/edit !> save()`
5. Activate/distribute, release lock	`wls:/wl_server/edit !> activate()`

Current Management Object

- CMO variable – current management object
  - Java object that serves as a proxy for direct access to the WLS MBean
  - Makes it easy to directly interact with the MBean – get and set attributes, other commands
  - Always set to the current WLST path
  - Only available for WLS MBeans, not custom MBeans
- Example:
  - wls:/mydomain/edit> cmo.setAdministrationPort(9092)
  - (This example changes the Administration Port in the Domain MBean)

Deploying an Application

- In Online mode, use the deploy command to deploy applications
  - Syntax: deploy(appName, path, [targets], [stageMode], [planPath], [options])
  - deploy("mainWebApp","C:/samples/server/examples/build/mainWebApp“,”server-1”)
  - Note: You do not need to be in an edit session to deploy applications.
- Reminder: in Offline mode, add an application using an extension template

Common Online Deployment Commands

- deploy Deploy an application to a WebLogic Server instance.
- distributeApplication Copy the deployment bundle to the specified targets.
- redeploy Redeploy a previously deployed application
- startApplication Start an application, making it available to users.
- stopApplication Stop an application, making it unavailable to users.
- undeploy Undeploy an application from the specified servers.
- updateApplication Updates an application configuration using a new deployment plan.

- Returns a WLSTProgressObject to track the progress of the command
  - You query the progress object to get the status; (pull, not push)

Interacting with the Node Manager

- You can use WLST to do the following with Node Manager:
  - Start a Node Manager.
  - Connect to a Node Manager, then use the Node Manager to start and stop servers on the machine on which Node Manager is running.

- Preferred method:
  - Use the Node Manager to start the Administration Server
  - Connect to the Admin Server
  - Start Managed Servers using the standard WLST lifecycle commands
  - Enables you to start all servers in the domain with one connection, regardless of which machines host the Managed Servers

Common WLST Node Manager Commands

- startNodeManager – starts Node Manager on the current machine.
- nm - Determines whether WLST is connected to Node Manager
- nmConnect - Connects WLST to Node Manager to establish a session. (Specify domain and credentials)
- nmDisconnect - Disconnects WLST from a Node Manager session.
- nmStart - Starts a server in the current domain using Node Manager.
- nmKill - Kills the specified server instance that was started with Node Manager.

Managing Server Lifecycle with WLST

- You can also manage server lifecycle through WLST without directly using Node Manager.
- The following WLST lifecycle commands are available:
  - startServer - Start the Administration Server. (Online or Offline)
  - start - Start a Managed Server instance or a cluster using Node Manager.
  - suspend - Suspend a running server.
  - resume - Resume a server instance that is suspended or in ADMIN state.
  - shutdown - Gracefully shut down a running server instance or cluster.
  - migrate - Migrate services to a target server within a cluster.

Admin Port & Side By Side Deployments

WebLogic Administration Port

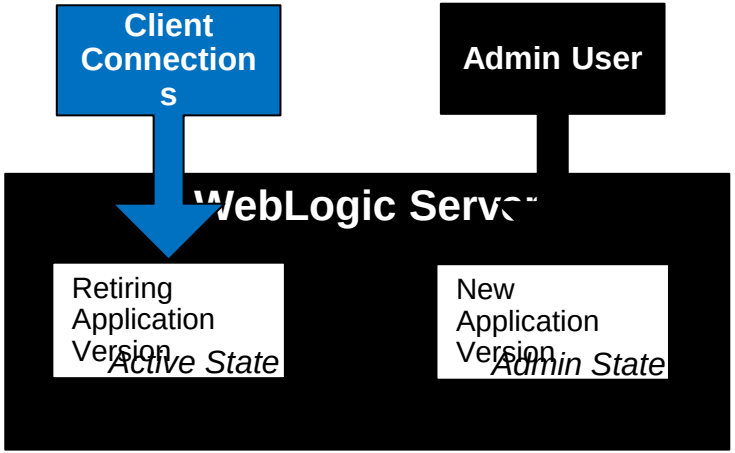
- Helps secure access to WebLogic Admin console

- By default WebLogic admin console is deployed on the same port as other applications
  - Adding “/console” to end of WebLogic domain gives you access

- With the administration port enabled:
  - The console is only accessible over a non-standard port. Admin requests over any port other than the admin port are rejected
  - You have to use SSL
  - You get a dedicated administration listen thread
  - It enables you to start a server in standby state
  - It enables you to separate administration traffic from application traffic in your domain

An Admin Port Use Case

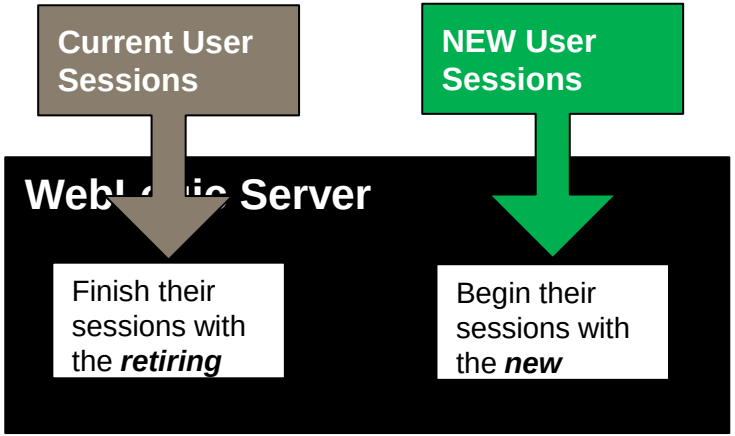
- On Production System – Test a new application or a new version of an existing application before opening it to everyone
- Deploy new application in admin state
- Application available only to Admin users
- Make sure the application works normally
- Promote the application and make it available to all users



Side-by-Side Deployment

- Two versions of the same application can be deployed side by side in WebLogic Server

- Users who currently have a session with the application can complete their session with the ‘retiring’ version of the application
- New users will have sessions created in the newly deployed version of the application
- Once all sessions are complete or expired from the retiring version it will be undeployed



Fast Swap

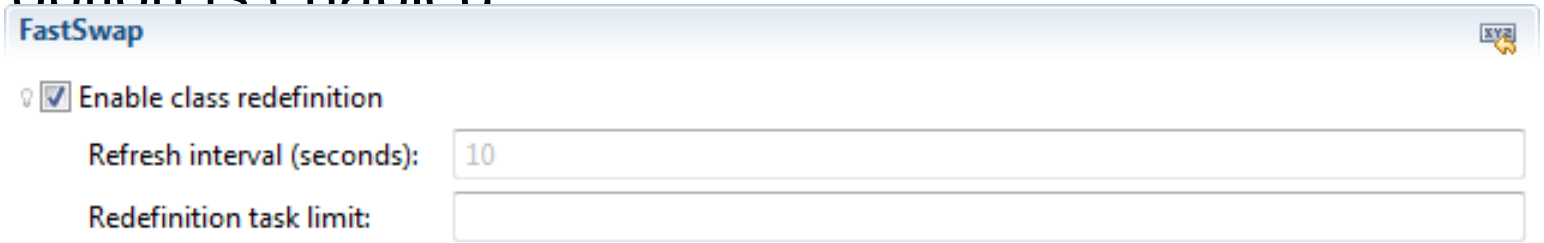
What is FastSwap? How does it improve Productivity?

- FastSwap uses
  - Java EE5's ability to redefine a class at runtime
  - Removes the restrictions on the shape of the class in weblogic

- FastSwap speeds up development cycles as
  - Java classes are now redefined in-place without reloading the classloader.
  - Developers can redeploy and navigate to wherever they were in the web page flow.

FastSwap & Eclipse

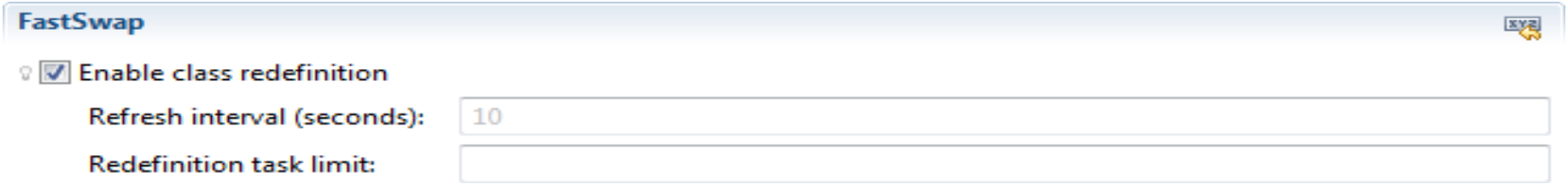
- Create a web app and an ejb module in an enterprise app from eclipse
- Changes to the session bean in the ejb module is available to the web app immediately, if the fastswap option is enabled



FastSwap deployment

- FastSwap uses
  - Redefines a class at runtime
  - Removes the restrictions on the shape of the class in WebLogic
- FastSwap speeds up development cycles as
  - Java classes are now redefined in-place without reloading the classloader.
  - Developers can redeploy and navigate to wherever they were in the web page flow.

Fastswap in eclipse (OEPE)



Maven

What is Maven?

- An Apache open source project
- Maven is:
  - An automated build system +
  - A project management system +
  - A library and dependency handling system +
  - A project description system +
  - A site generation system +
  - …
- Mature
  - Maven 1.0 2004
  - Maven 2.0 2005
  - Maven 3.0 end 2010

maven

Maven is a project management tool which encompasses a project object model, a set of standards, a project lifecycle, a dependency management system, and logic for executing plugin goals at defined phases in a lifecycle.
When you use Maven, you describe your project using a well-defined project object model, Maven can then apply cross-cutting logic from a set of shared (or custom) plugins.

Jason Van Zyl, Sonatype

WebLogic Server Maven Support Overview

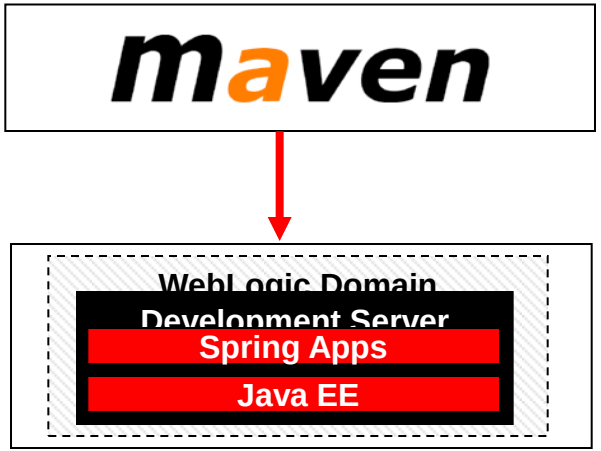
- Introduced with WebLogic Server 11g R1 PS3 (10.3.4) supporting Application Deployment operations
  - Maven Mojo + WebLogic Deployer + WebLogic Client
  - Supported Deployment Lifecycle operations: list-apps, deploy/undeploy, start, stop and update

- WebLogic 12c (12.1.1) provides additional functionality since the 11g release
  - Installation of Weblogic ZIP Distribution onto a machine where WebLogic has not been installed
  - WebLogic Domain Creation
  - Start/Stop WebLogic Servers
  - Execute WLST Scripts

WebLogic 12.1.1 Maven Integration

New Support for WebLogic Server Lifecycle Management

- Enable/support/automate the full development lifecycle from Maven
- New Maven goals/functional support
  - install: automate install from zip file
  - domain creation: create a simple domain
  - server lifecycle: start-server, stop-server
  - wlst: execute inline and external WLST scripts
  - appc: compile Maven artifacts using appc
  - deployment: deploy apps to server
- Simple and intuitive
- Additional WebLogic/FMW enhancements planned



Configure WebLogic Maven Plugin in Project

Maven pom.xml Configuration

- WebLogic Maven Plugin needs to be configured as a plugin for a project to enable it to be used
  <project> <build> <plugins> <plugin>
- Specify WebLogic Server Credentials – defaults can be used for the rest!

```
<build>
  <plugins>
    <plugin>
      <groupId>com.oracle.weblogic</groupId>
      <artifactId>wls-maven-plugin</artifactId>
      <version>12.1.1.0</version>
      <configuration>
        <user>weblogic</user>
        <password>welcome1</password>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Parameter	Default Value
adminURL	t3://127.0.0.1:7001
name	${project.build.finalName}
source	${project.build.directory}/ ${project.build.finalName}.${project.packaging}
domainHome	./Oracle/domains/mydomain
middlewareHome	./Oracle/Software

Installing the WebLogic Maven Plugin

WebLogic-Maven Plugin

- Use Maven install:install-file goal to install the wls-maven-plugin.jar library
- Use $WL_HOME/server/lib/pom.xml to specify Group ID, Artifact ID and Version
- The install:install-file will install the plugin into your local repository
- Alternatively deploy:deploy-file can be used to deploy the plugin to a remote repository

```
$ mvn install:install-file –Dfile=$WL_HOME/server/lib/wls-maven-plugin.jar
                           –DpomFile=$WL_HOME/server/lib/pom.xml
```

Deploying Applications with Maven

WebLogic-Maven Plugin

- Deploy applications after they are packaged. Defaults:
  - Application Name: ${project.build.finalName} (ArtifactId-Version)
  - Source: ${project.build.directory}/${project.build.finalName}.${project.packaging}
  - Target: AdminServer
- Common weblogic.Deployer properties used:
  - stage / nostage
  - remote
  - upload
  - userConfigFile / userKeyFile (for obfuscating the login password)

```
$ mvn wls:deploy
```

Re-Deploying Applications with Maven

WebLogic-Maven Plugin

- Redeploy already deployed applications to same or new targets
- Key Defaults:
  - Application Name: ${project.build.finalName} (ArtifactId-Version)
  - Target: AdminServer

```
$ mvn wls:redeploy
```

Un-Deploying Applications with Maven

WebLogic-Maven Plugin

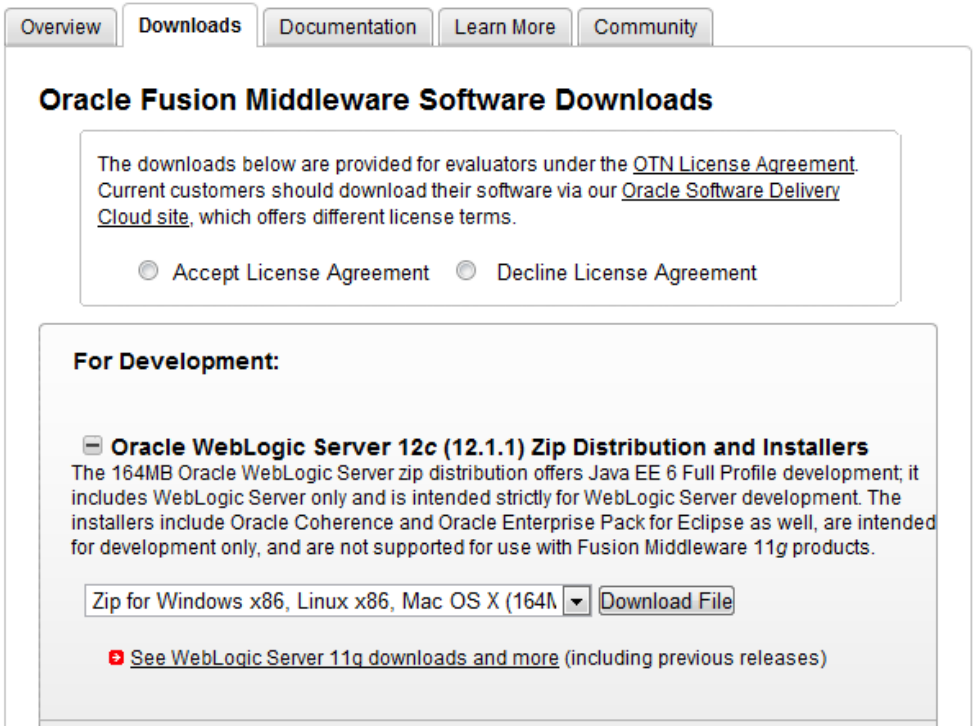
- Un-Deploy applications
- Key Defaults:
  - Application Name: ${project.build.finalName} (ArtifactId-Version)
- Common weblogic.Deployer parameters:
  - timeout
  - verbose
  - graceful

```
$ mvn wls:undeploy
```

WebLogic Zip Distribution & WLS:INSTALL goal

164MB WebLogic Distribution

- The WebLogic Zip Distribution can be downloaded from OTN
- The WebLogic Zip distribution is required to use the wls:install goal
- Can be made available via:
  - A local/remote Maven Repo
  - A filesystem
  - An HTTP URL



Installing the WebLogic Zip Distribution into your Local Repository

- The WebLogic ZIP Distribution must be available in your local repository or a remote repository
- Install the WebLogic ZIP Distribution into your local repository using the Maven install:install-file goal

```
$ mvn install:install-file -Dfile=wls1211_dev.zip
                           -DgroupId=com.oracle.weblogic
                           -DartifactId=wls-dev
                           -Dpackaging=zip
                           -Dversion=12.1.1.0
```

Installing WebLogic using the ZIP Distribution

WebLogic-Maven Plugin

- Install WebLogic Zip Distribution using:
  - A Maven repository (com.oracle.weblogic:wls-dev:zip:12.1.1.0)
  - An HTTP URL
  - A filesystem path
- Key Defaults:
  - Middleware Home: ${project.directory}/Oracle/Software

```
$ mvn wls:install –DartifactLocation=[Maven Artifact | HTTP URL | File Path]
```

Creating a WebLogic Domain

WebLogic-Maven Plugin

- Create a WebLogic Domain
  - Using the default template
  - OR Specify a custom template
- Key Defaults
  - Domain Home: ${project.directory}/Oracle/Domains/myDomain
  - Domain Template: none – default Domain with Admin Server

```
$ mvn wls:create-domain –
DdomainTemplate=src/main/wls/myTempalte.jar
                                –DdomainHome=target/domain
```

Start and Stop a WebLogic Server Instance

WebLogic-Maven Plugin

- Starts or Stops the AdminServer for a WebLogic Domain
  - Can Start/Stop Managed Servers by overriding the ‘command’ parameter
- Key Defaults:
  - Domain Home: ${project.directory}/Oracle/Domains/myDomain
  - Middleware Home: ${project.directory}/Oracle/Software

```
$ mvn wls:start-server
```

```
$ mvn wls:stop-server
```

Execute WLST using Maven

WebLogic-Maven Plugin

- Execute a WLST script
  - propertiesFile – optional Java properties file passed as command-line Java parameters
  - Online or Offline
- Execute multiple scripts using multiple plugin declarations or commands

```
$ mvn wls:wlst –DfileName=src/main/wlst/createDatasource.py
                –
DpropertiesFile=src/main/resources/dev.properties
```

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Hardware and Software

ORACLE

Engineered to Work Together