Oracle® Fusion Middleware

Administering Zero Downtime Patching Workflows

14c (14.1.2.0.0)
F69890-01
December 2024

Oracle Fusion Middleware Administering Zero Downtime Patching Workflows, 14c (14.1.2.0.0)

F69890-01

Contents

Preface

Audience v
Documentation Accessibility v
Diversity and Inclusion v
Related Documents vi
Conventions vi

1 Introduction to Zero Downtime Patching

What Is Zero Downtime Patching? 1-1
Identifying a Zero Downtime Patch 1-2
Types of Patching Workflows 1-2
The Patching Workflow Process 1-3
Reverting an Update 1-3
Rolling Out a Patched Oracle Home: Overview 1-4
Rolling Out a New Java Version: Overview 1-5
Rolling Out Updated Applications: Overview 1-6
In-Memory Session Replication for ZDT Rollouts 1-8

2 Preparing for Zero Downtime Patching

ZDT Patching Restrictions 2-1
Preparing to Migrate Singleton Services 2-2
    Creating a JSON File for Migrating Singleton Services 2-4
Preparing to Roll Out a Patched Oracle Home 2-6
    Creating a Second Oracle Home 2-6
    Applying Patches to the Second Oracle Home 2-7
    Creating an Archive and Distributing It to Each Node 2-7
Preparing to Upgrade to a New Java Version 2-8
Preparing to Update to New Application Versions 2-9
    The Effects of Staging Modes 2-9
    Creating an Application Update JSON File 2-10

3 Patching an Existing WebLogic Server Installation

Using Zero Downtime Patching 3-1
Obtaining a List of Applied Patches 3-2

4 Configuring and Monitoring Workflows

Strategies for Rolling Out a Patched Oracle Home 4-1
Starting the Administration Server 4-2
Using WLST to Initiate and Monitor Workflows 4-3
    Rolling Out a New Oracle Home 4-7
    Updating Your Java Version 4-8
    Updating Both Oracle Home and the Java Version 4-9
    Rolling Out Updated Applications 4-9
    Reverting to the Previous Oracle Home, Java Home, or Applications 4-10
    Initiating a Rolling Restart of Servers 4-10
    Monitoring Workflow Progress 4-11
    Running, Reverting, and Resuming Stopped Workflows 4-12
    Useful WLST Commands for Workflows 4-13
    Sample WLST Script 4-13

5 Modifying Workflows Using Custom Hooks

About Extension Points 5-1
The Patching Workflow Process for Custom Hooks 5-3
Specifying Extensions to Modify the Workflow 5-4

Preface

This document, *Administering Zero Downtime Patching Workflows*, describes how to move a domain from an existing Oracle home to a patched Oracle home, update to a new Java version, or update applications in a domain without any loss of service. It describes how to create workflows that methodically apply the changes to the servers in the domain while keeping the domain available. It also describes how to monitor the progress of workflow tasks and revert the domain to its previous state.

- Audience
- Documentation Accessibility
- Diversity and Inclusion
- Related Documents
- Conventions

Audience

This document is written for WebLogic Server administrators and operators who are responsible for applying updates to a domain, such as Oracle patches to an Oracle home, new Java versions, or application updates. It is assumed that readers are familiar with the WebLogic Server Administration Console, WebLogic Scripting Tool (WLST), and the operating system and platform on which Oracle WebLogic Server is installed.

Documentation Accessibility

Diversity and Inclusion

Related Documents

See the following Oracle Fusion Middleware documents:

- *Patching with OPatch*
- *Administering Node Manager for Oracle WebLogic Server*
- *Understanding the WebLogic Scripting Tool*
- *WLST Command Reference for WebLogic Server*
- *Deploying Applications to Oracle WebLogic Server*
- *MBean Reference for Oracle WebLogic Server*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Introduction to Zero Downtime Patching

Zero Downtime Patching in Oracle WebLogic Server provides a way to create various workflows to apply updates across a domain without interrupting your applications to service requests. Learn how to apply an update across a domain, revert an update, and understand the workflow process.

- What Is Zero Downtime Patching?
- Identifying a Zero Downtime Patch
  You can identify a ZDT patch by the value of the patch uptime option in the patch metadata.
- Types of Patching Workflows
  You can create different types of workflows with ZDT Patching, for moving servers to a patched Oracle home, updating to a new Java version, deploying updated applications, and more.
- The Patching Workflow Process
  A ZDT Patching workflow constitutes a systematic set of steps that are run in a particular order to roll out an update.
- Reverting an Update
  During the process of the patching workflow, ZDT Patching monitors the success and failure of each step and provides the capability to revert to the previous step. You can configure the revert process to run automatically, or initiate the process manually.
- Rolling Out a Patched Oracle Home: Overview
  You can roll out a patched Oracle home across your domain by using WLST while ensuring that your application continues servicing requests.
- Rolling Out a New Java Version: Overview
  You can roll out a new Java version across your domain without affecting the continuity of servicing requests during the patching process. Use WLST to roll out updates to Java home.
- Rolling Out Updated Applications: Overview
  ZDT provides the ability to update applications deployed to your domain without causing the application to suffer downtime. Use WLST to roll out application updates.
- In-Memory Session Replication for ZDT Rollouts
  During ZDT rollouts, the forceful shutdown of a server could lead to loss of in-memory sessions. To avoid any loss of session data, set the `rollout` command to allow time for the graceful shutdown of the server instance before shutting it down forcefully.

What Is Zero Downtime Patching?

Zero Downtime Patching (ZDT Patching) automates the rollout of out-of-place patching or updates across a domain while allowing your applications to continue servicing requests. After defining your patching strategy, you can use the WebLogic Scripting Tool (WLST) to orchestrate the rollout of updates across some or all the servers in your domain.
Although WebLogic Server has supported rolling upgrades since version 9.2, the process has always been manual. ZDT Patching automates this process by using workflows that you define. You can patch or update any number of nodes in a domain with little or no manual

intervention. Changes are rolled out to one node at a time, allowing a load balancer, such as Oracle Traffic Director, to redirect incoming traffic to the remaining nodes until the node has been updated.

ZDT Patching also provides support for custom hooks. The ZDT custom hooks provide a flexible mechanism for modifying the patching workflow by running additional scripts at specific points in the patching rollout. This feature allows application developers and administrators to include any operation that is specific to a particular type of rollout but that may not be applicable to the base patching workflow. For more information about using this feature, see Preparing to Modify Rollouts Using Custom Hooks.

Identifying a Zero Downtime Patch

You can identify a ZDT patch by the value of the patch uptime option in the patch metadata.

After you download a patch, open up `patchdeploy.xml` in the `*PATCH_HOME*/etc/config` directory, where *PATCH_HOME* is the location of the patch directory that contains the patch. If the value of `patch-uptime-option` is `FMW_ROLLING_ORACLE_HOME`, as shown in the following example:

```
<patch-uptime-option>FMW_ROLLING_ORACLE_HOME<patch-uptime-option>
```

Or the value is `FMW_ROLLING_SESSION`:

```
<patch-uptime-option>FMW_ROLLING_SESSION<patch-uptime-option>
```

Then, the patch is suitable for ZDT patching.

If `FMW_ROLLING_ORACLE_HOME` or `FMW_ROLLING_SESSION` does not appear in the patch metadata, then you know that the patch is *not* suitable for ZDT patching. As a result, the patch is not compatible with a ZDT patch plan.

Types of Patching Workflows

You can create different types of workflows with ZDT Patching, for moving servers to a patched Oracle home, updating to a new Java version, deploying updated applications, and more.

You can create a workflow that performs any one of these tasks. You also can create a workflow that performs any combination of an Oracle home update, Java version update, and application update.

- **Moving servers to a patched Oracle home**: The workflow transitions the Administration Server or clusters or both, to another Oracle home that already has been patched using the OPatch utility.
- **Updating to a new Java version**: The workflow updates the Administration Server or clusters or both, to use a newly installed Java home.
- **Deploying updated applications**: The workflow deploys updated applications to the selected clusters.
- **Performing a rolling restart of servers**: The workflow sequentially restarts the Administration Server or servers in the selected clusters or both safely, including the graceful shutdown of the servers and starting them up again.

Prior to creating a patching workflow, you must complete the preliminary steps for each of these tasks with the exception of rolling restarts. See Preparing for Zero Downtime Patching.

The Patching Workflow Process

A ZDT Patching workflow constitutes a systematic set of steps that are run in a particular order to roll out an update.

When you use a ZDT patching workflow to roll out an update, the rollout:

- Systematically works its way through each applicable node
- Identifies the servers on the node that are included in the rollout
- Gracefully shuts down those servers
- When switching to a patched Oracle home:
  - Backs up the existing Oracle home to a backup directory
  - Calls Node Manager to switch the contents of the current Oracle home to the contents of the specified Oracle home
- When updating to a new Java version:
  - Updates all scripts in the domain's Oracle home that contain a reference to Java home to point to the new Java home
  - Updates all scripts in the domain's home directory that contain a reference to Java home to point to the new Java home
- When updating to new application versions:
  - Locates the current directory for each application
  - Moves the current directory for each application to a backup location
  - Moves the directory for the new version of each application to the location of each original application
- Restarts each server once the update has completed on the node

The workflow runs the appropriate steps in order and monitors the success of each step. If a step fails, the workflow may attempt to retry it. If a step cannot be completed successfully, then the workflow reverts each previous step in order. Updates can be reverted either automatically or can be initiated manually, as described in Reverting an Update.

Reverting an Update

During the process of the patching workflow, ZDT Patching monitors the success and failure of each step and provides the capability to revert to the previous step. You can configure the revert process to run automatically, or initiate the process manually.

ZDT Patching is able to revert an update at any point in the process, even after it has completed. Updates can be reverted:

- **Automatically**—When creating a workflow, you can opt to have the update revert automatically if there is a failure. The update will be rolled back from the point of failure, starting with the last successfully completed step.
- **Manually**—While a workflow is in progress, you can stop it and revert the process at any point. The update will then be rolled back, starting with the last successfully completed step.

  After a workflow has completed, you can create a workflow to reverse the update that was made. The revert process differs slightly depending on the update. If you are reverting to the previous Oracle home, then you are provided with an option to specify that the process is a rollback. For Java and applications, to revert you can point to the previous version of Java or the application.

For information about reverting an update, see Running, Reverting, and Resuming Stopped Workflows.

Rolling Out a Patched Oracle Home: Overview

You can roll out a patched Oracle home across your domain by using WLST while ensuring that your application continues servicing requests.

> **Note:**
>
> OPatchAutoFMW (installed in `OPatch/auto/fmw` directory) is deprecated and is automatically removed when you update to OPatch 13.9.4.2.2 or later. You can continue to use OPatchAutoCore (installed in `OPatch/auto/core` directory) for auto updates during Fusion Middleware installation.

Before rolling out a patched Oracle home to all nodes in your domain, ensure that the following conditions are met:

- The domain is distributed across all nodes and stored in the same location on all nodes.
- The existing Oracle home is in the same location on all nodes.
- Node Manager is running on all nodes.
- All Managed Servers in all clusters that will be included in the rollout are running.

See ZDT Patching Restrictions, for additional requirements and restrictions. Figure 1-1 shows the sequence of operations that are performed for an Oracle home rollout on each node, regardless of how you perform the rollout.

To roll out a patched Oracle home, perform the following tasks:

1. Create and distribute the patched Oracle home archive.

   Manually create and distribute the patched Oracle home archive:

   a. Use the `copyBinary` command to create an archive of your existing Oracle home.

      For details on this step and the next step, see Creating a Second Oracle Home.

   b. Use the `pasteBinary` command to create an Oracle home to be patched on a development or test system that has a domain topology similar to your production domain. This gives you an Oracle home that has the same patch level and products as you have on your production system.

   c. Use OPatch to apply the desired patch or patches to the Oracle home on your development or test system.

      See Applying Patches to the Second Oracle Home.

   d. Test and verify the patched Oracle home.

   e. When you are satisfied that the patched Oracle home is stable, use `copyBinary` to create an archive of the patched Oracle home.

      For details on this and the next step, see Creating an Archive and Distributing It to Each Node.

f. Distribute this archive to all nodes in your production system.

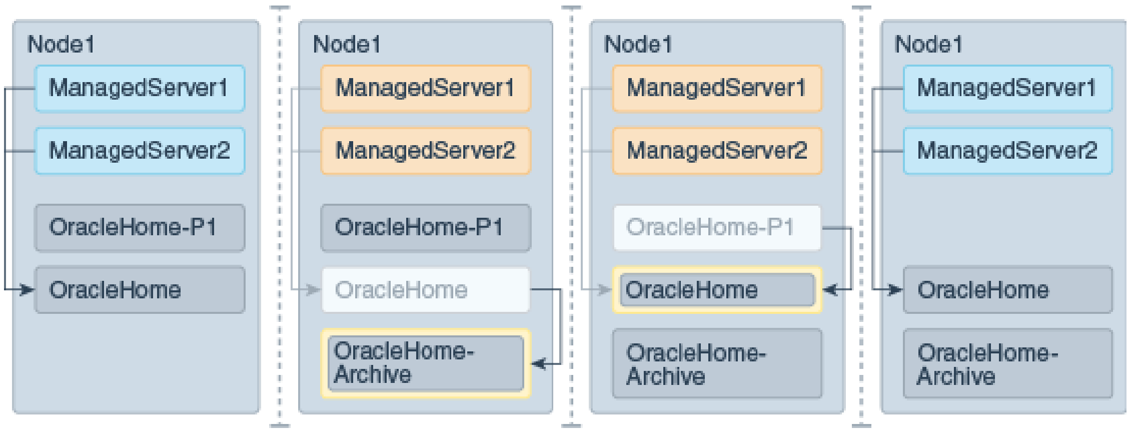
> **Note:**
>
> There is no need to use `pasteBinary` to create the archive on each node. The rollout process will create the new Oracle home on each node from the archive.

2. Create a ZDT workflow to roll out the patched Oracle home to your Administration Server. You can do this by using the WLST `rolloutOracleHome` command and specifying the Administration Server as the rollout target. See Rolling Out a New Oracle Home.
3. After the workflow completes successfully, create another ZDT workflow to roll out the patched Oracle home to the clusters in your domain. You can do this by using the WLST `rolloutOracleHome` command and specifying a comma-separated list of clusters as the rollout target.

> **Note:**
>
> You can combine the last two steps into one workflow by specifying the domain as the target in the `rolloutOracleHome` command.

Figure 1-1 Oracle Home Rollout Operations

This figure shows the sequence of operations that are performed for an Oracle home rollout on each node, regardless of how you perform the rollout.



Rolling Out a New Java Version: Overview

You can roll out a new Java version across your domain without affecting the continuity of servicing requests during the patching process. Use WLST to roll out updates to Java home.

Before rolling out a new Java version to all nodes in your domain, ensure that the following conditions are met:

- The domain is distributed across all nodes and is stored in the same location on all nodes.

- Oracle home must be in the same location on all nodes.
- Node Manager is running on all nodes.
- All Managed Servers in all clusters that will be included in the rollout is running.

See ZDT Patching Restrictions, for additional requirements and restrictions.

To roll out a new Java version:

1. Install the new Java version on all nodes. The full path to this Java home must be the same on all nodes.

   See Preparing to Upgrade to a New Java Version.
2. Create a ZDT workflow to roll out the new Java home to your Administration Server. You can do this by using the WLST `rolloutJavaHome` command and specify the Administration Server as the rollout target. See Updating Your Java Version.
3. After the workflow completes successfully, create another ZDT workflow to roll out the new Java home to the clusters in your domain. To do this, use the WLST `rolloutJavaHome` command and specify a comma-separated list of clusters as the rollout target.

> **Note:**
>
> You can combine the last two steps into one workflow by specifying the domain as the target in the `rolloutJavaHome` command.

Rolling Out Updated Applications: Overview

ZDT provides the ability to update applications deployed to your domain without causing the application to suffer downtime. Use WLST to roll out application updates.

This section provides an overview of how to roll out new application versions to Managed Server nodes in your domain.

Prior to doing the rollout, ensure that the following conditions are met:

- The domain that is being updated is distributed across all nodes and must be stored in the same location on all nodes.
- Oracle Home is in the same location on all nodes.
- Node Manager is running on all nodes.
- All Managed Servers in all clusters that will be included in the rollout is running.

> **Note:**
>
> WebLogic Server does not support the rollout of applications deployed to the Administration Server. Applications deployed to the Administration Server cannot be updated without downtime because session replication can be applied only to clustered instances, whereas Administration Server is a standalone instance.

See ZDT Patching Restrictions, for additional requirements and restrictions. The figures in this section illustrate the scenario for patching staged, no-stage, and external staged applications.

During the rollout, the patched application source will be moved to the appropriate application source location for each stage type.

To roll out new application versions to your Managed Servers:

1. Place a copy of the updated application directory as follows:
   - (Stage mode) Place a copy of each updated application directory on the domain's Administration Server.
   - (No-stage mode and external stage mode) Place a copy of each updated application directory on each node that will be affected. The directory must be the same on each node.

   See The Effects of Staging Modes.
2. Create a JavaScript Object Notation (JSON) file that defines each application name, the path and file name for each updated application archive, and the path and file to which you want to back up the original application archive.

   See Creating an Application Update JSON File.
3. Create a ZDT workflow to roll out the new application versions. To do this, use the WLST `rolloutApplications` command and specify a comma-separated list of clusters as the rollout target.

Figure 1-2 Patching Staged Applications

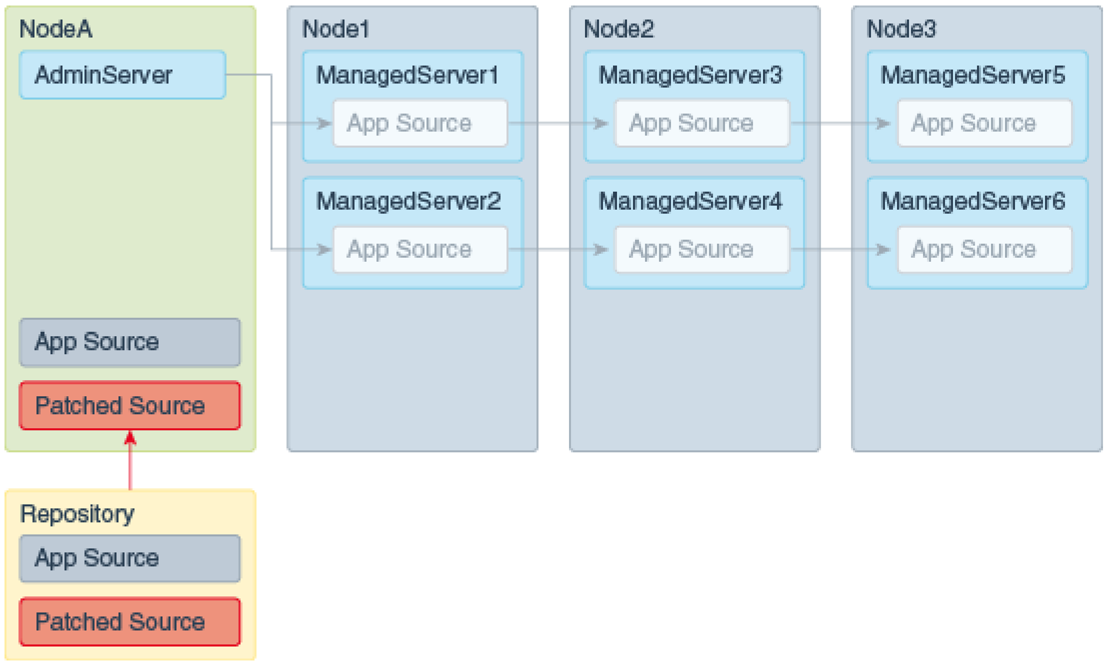


Figure 1-3 Patching No-Stage Applications

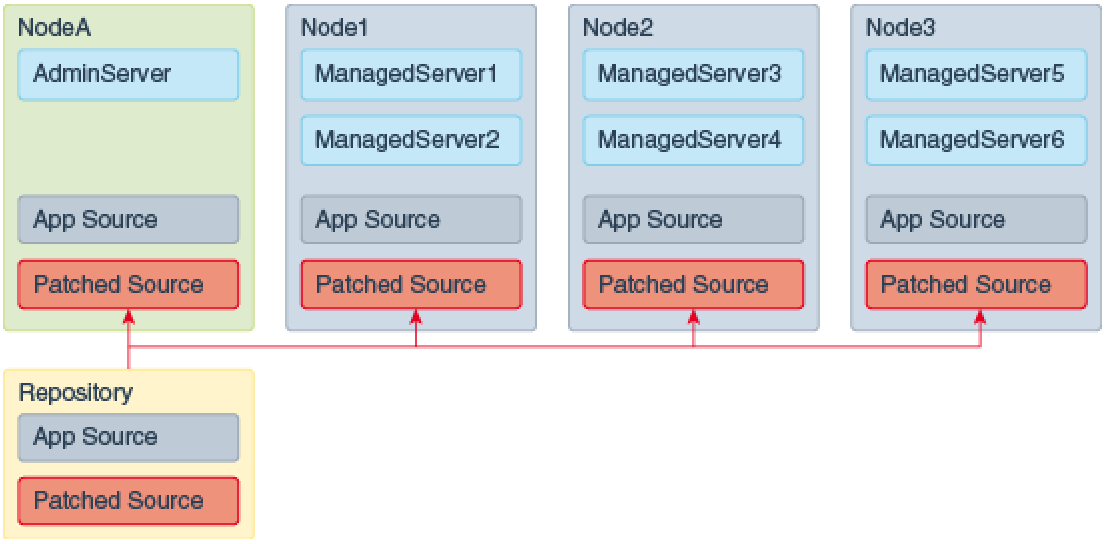
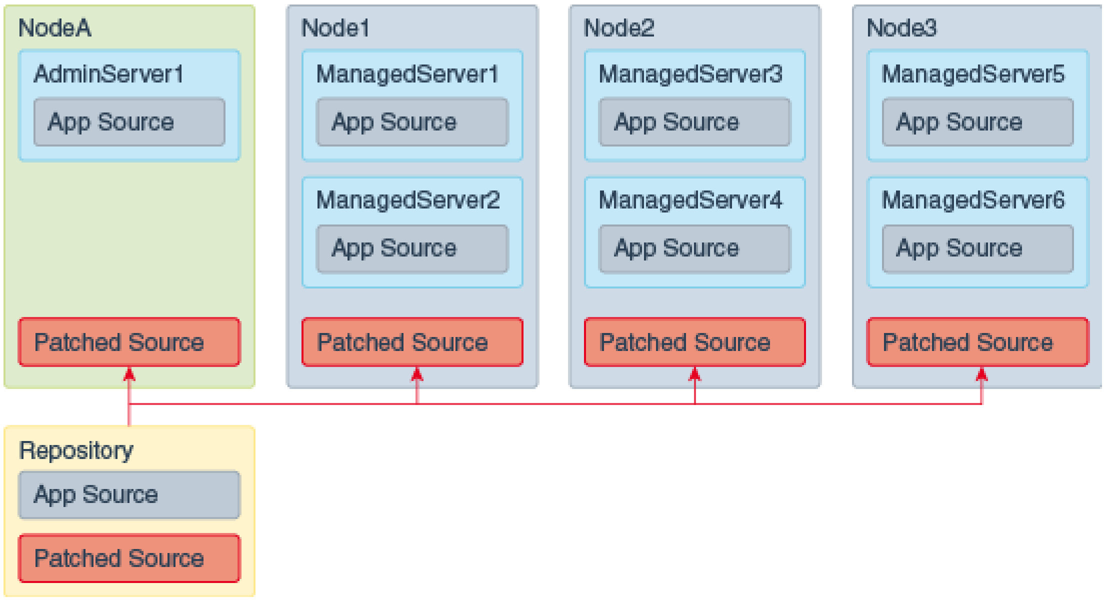


Figure 1-4 Patching External Staged Applications



In-Memory Session Replication for ZDT Rollouts

During ZDT rollouts, the forceful shutdown of a server could lead to loss of in-memory sessions. To avoid any loss of session data, set the `rollout` command to allow time for the graceful shutdown of the server instance before shutting it down forcefully.

For web applications that use in-memory session replication, the in-memory sessions are never replicated or persisted to allow for failover. As a result web applications may lose session state due to sudden failure of a server or front-end misdirection causing the request to land on a server without the session.

With regard to Zero Downtime (ZDT) rollouts, when you shut down any server that holds the in-memory session, the server waits for that session to complete before shutting down. Because the default value for session timeout is 1 hour, the server may be in the SUSPENDING state for 1 hour or even longer if sessions continue to be used or updated. If you do not wait for the session to complete its life cycle, then the state is lost because in-memory sessions are neither replicated nor persisted for web applications.

If you do not want to wait for an hour or longer, then Oracle recommends that you set the `shutdownTimeout` argument in the WLST `rollout`command to the time (in seconds) that you want the server to wait before shutting down. For information about using the `shutdownTimeout` argument, see Table 4-1.

2

Preparing for Zero Downtime Patching

Before configuring a patching workflow, ensure that you perform the required preliminary steps such as installing and patching a new Oracle home, installing a new Java version, or installing updated applications on each node. There are also known restrictions to consider before preparing for and creating a ZDT patching workflow in Oracle WebLogic Server.

- ZDT Patching Restrictions
  For the rollout orchestration to be successful, you must keep in mind certain restrictions before you configure a patching workflow.
- Preparing to Migrate Singleton Services
  ZDT Patching rollouts provide support to migrate singleton services, such as JMS and JTA, using the service migration feature of WebLogic Server. For better control of service migration during a rollout, you can also use the JSON file-based migration option that ZDT supports.
- Preparing to Roll Out a Patched Oracle Home
  Before rolling out a patched Oracle home to your Managed Servers, you must create an Oracle home archive and distribute it to each node.
- Preparing to Upgrade to a New Java Version
  Before upgrading to a new Java version, you must copy the new Java version to each node you want to include in the upgrade. Before installing the new Java version, there are certain conditions that must be met.
- Preparing to Update to New Application Versions
  Before rolling out an application update, the new application version is distributed to all affected nodes depending on the staging mode you used when you staged the application. You must create a JSON file to specify the properties of applications that require an update.

ZDT Patching Restrictions

For the rollout orchestration to be successful, you must keep in mind certain restrictions before you configure a patching workflow.

Prior to preparing for and creating a ZDT patching workflow, consider the following restrictions:

- The Managed Servers that are included in the workflow must be part of a cluster, and the cluster must span two or more nodes.
- If you want to roll out an update to the Managed Servers without targeting and updating the Administrations Server, then ensure that the Administration Server is on a different node than any of the Managed Servers being updated.
- If you are updating to a patched Oracle home, the current Oracle home must be installed locally on each node that will be included in the workflow. Although it is not required, Oracle also recommends that the Oracle home be in the same location on each node.
- When you are rolling out a new Oracle home using WLST commands, you must specify the path to the JAR archive that contains the Oracle home to roll out. Specifying a local directory is not supported when you are rolling out a new Oracle home. `Only` if you are rolling back to a previous Oracle home, you can specify the path to the local directory

which must be the backup Oracle home directory from the previous rollout that you want to roll back to.

- If Managed Servers on a node belong to different clusters and those clusters share the same Oracle home, then if you include one of those clusters in a workflow, you must also include the other cluster in the workflow. For example, if Node 1 has Managed Server 1 in Cluster 1 and Managed Server 2 in Cluster 2, and both Cluster 1 and Cluster 2 share the same Oracle home, then if you include Cluster 1 in the workflow, you must also include Cluster 2. This applies to Java home, Oracle home and application update rollouts.
- The domain directory must reside outside of the Oracle home directory.
- (Windows only) When you use the WebLogic Scripting Tool (WLST) to initiate a rollout of a new Oracle home, you cannot run WLST from any Oracle home that will be updated as part of the workflow. Instead, use one of the following options:
  - Run WLST from an Oracle home on a node that will not be included in the workflow. This Oracle home must be the same version as the Oracle home that is being updated on other nodes.
  - Run WLST from another Oracle home that is not part of the domain being updated. This Oracle home must be the same version as the Oracle home that is being updated. It can reside on any node, including the Administration Server node for the domain being updated.
- (Windows only) Windows file locks may pose problems during the ZDT rollout operations. You must attempt to rectify these common file handle lock issues before running a rollout on Windows to avoid rollout failure:
  - Using the WLST client on the Administration Server will cause the Oracle home directory to be locked. This will cause any rollout on that node, including a domain rollout to fail. To avoid this, use a WLST client installed on a node that is not targeted by the rollout.
  - Opening command terminals or applications residing in any directory under Oracle home may cause a file lock. As a result, you will be unable to update that particular Oracle home.
  - Any command terminal or application that references the application source file or a JAR file may cause a file lock, making it impossible to update that particular application.

Preparing to Migrate Singleton Services

ZDT Patching rollouts provide support to migrate singleton services, such as JMS and JTA, using the service migration feature of WebLogic Server. For better control of service migration during a rollout, you can also use the JSON file-based migration option that ZDT supports.

All ZDT rollouts require a restart of the servers that are included in the rollout. One feature of the rollout is detection and handling of singleton services, such as Java Transaction API (JTA) and Java Messaging Service (JMS). To make these singleton services highly available during the rollout operation, ZDT patching takes advantage of the service migration mechanisms supported by WebLogic Server. For singleton services in your environment, service migration can be configured in either of the following ways:

- For migrating a singleton service that is configured using migratable targets, the service migration is configured as described in Service Migration in *Administering Clusters for Oracle WebLogic Server*. If a service is configured using migratable targets and the migration policy is set to `exactly-once`, then the service automatically migrates during the graceful shutdown of a server. If, however, the migration policy for a service is `manual` or `failure-recovery`, then you must take steps to ensure that the service is migrated safely during server shutdown. To achieve this, you must define the migration properties in the JSON file as described in Creating a JSON File for Migrating Singleton Services.

You must bear in mind the following issues restrictions when migrating singleton services that is configured using migratable targets:

  – The data store for JMS servers must reside at a shared location to be used by the members of the cluster, without which the user might experience loss of messages.
  – The ClusterMBean must be configured with the `setServiceActivationRequestResponseTimeout` method and its value must be set depending on the time taken for the migration to succeed.
  – The JNDI NameNotFoundException is returned during lookup for JMS connection factories and destinations. This is a known limitation. For information about this limitation and its workaround, see note 1556832.1 at My Oracle Support.
  – As services migrate during the rollout, the JNDI lookup for JMS connection factories and destinations fail. In such cases of server failure, JMS applications attempt to reconnect to another available server for non-deterministic time till the migration succeeds. See Recovering from a Server Failure in *Developing JMS Applications for Oracle WebLogic Server*.

- For migrating a singleton service that is configured using the JMS cluster configuration, the service migration is configured (depending on your cluster type) as described in Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*. If a service is configured using the JMS Cluster configuration, then the migration-policy must be set to `Always` to enable the automatic migration of services during the graceful shutdown of a server. If the migration-policy is `On-Failure` or `Off`, then you must take steps to ensure that the service is migrated safely during server shutdown. You must also ensure that the automatic `restart-in-place` option is explicitly disabled when using this simplified HA service migration model.

  .

Note:

ZDT rollout allows you to specify whether a singleton service should be migrated before shutting down during patching. However, during the rollout operation, the user is not allowed to specify the migration of servers on the same machine. This is because, all servers on a machine experience shutdown during a rollout which may cause unavoidable downtime for users. Ensure that you always specify migration of services to a server on a different machine, failing which the rollout might fail.

Service migration involves shutting down one or more singleton services on the first server that is being rolled out. This means that the service is made available on the second server while rollout is in progress. Upon successful completion of the rollout, the services are migrated back to the newly patched first server. Since this process involves restarting of singleton services, the users can expect a brief downtime of services when the service is shut down on the first server and has not fully started on the second server. This would render the service unavailable and applications may experience a brief outage. The period of downtime of services may depend on factors including, hardware (both machine and network) performance, cluster size, the server startup time, and persistent message backlog in case of JMS.

- Creating a JSON File for Migrating Singleton Services

Creating a JSON File for Migrating Singleton Services

To ensure that the singleton service is migrated safely during server shutdown, you must perform the following tasks:

- Create a JSON file to define migration properties for such services, as described in this section
- Configure the rollout to use the JSON file as described in Configuring and Monitoring Workflows.

The JSON file must start with the following line:

```
{"migrations":[
```

Each singleton service migration that you need to migrate is defined using the parameters described in the following table.

Parameter	Description
`source`	The name of the source server from which the service is to be migrated. This parameter is required.
`destination`	For `migrationType` of `jms`, `jta`, or `all`, the name of the destination server to which the service is to be migrated. For `migrationType` of `server`, the name of another machine (node) in the domain on which Node Manager is running. This parameter is required if the `migrationType` is `jms`, `jta`, `server`, or `all`.
`migrationType`	The type of migration, which can be one of the following types: • `jms` — Migrate all JMS migratable targets from the source server to the destination server. • `jta` — Migrate all JTA services from the source server to the destination server. • `server` — Invoke Whole Server Migration to perform a server migration. The destination must be a machine (node) on which Node Manager is running. • `all` — Migrate all services (for example, JTA and JMS) from the source server to the destination server. • `none` — Disable service migration from the source server. If you specify this type, `failback` and `destination` are not needed.

Parameter	Description
`failback`	If set to `true`, a failback operation is performed. Failback restores a service to its original hosting server, the server on which it was running before the rollout. The default value is `false` (no failback). **Note:** A JTA service automatically fails back when it is invoked for migration. Therefore, do not use the `failback` option for JTA services, as it does not apply to them. The rollout fails if you specify the `failback` option.

The following sample JSON file shows how to define various migration scenarios.

```
    {"migrations":[

# Migrate all JMS migratable targets on server1 to server2. Perform a failback
    {
    "source":"server1",
    "destination":"server2",
    "migrationType":"jms",
    "failback":"true"
    },

# Migrate only JTA services from server1 to server3. Note that JTA migration
# does not support the failback option, as it is not needed.
    {
    "source":"server1",
    "destination":"server3",
    "migrationType":"jta"
    },

# Disable all migrations from server2
    {
    "source":"server2",
    "migrationType":"none"
    },
    {

# Migrate all services (for example, JTA and JMS) from server 3 to server1 with
# no failback
    "source":"server3",
    "destination":"server1",
    "migrationType":"all"
    },

# Use Whole Server Migration to migrate server4 to the node named machine 5 with
# no failback
    {
    "source":"server4",
    "destination":"machine5",
    "migrationType":"server"
    }

    ]}
```

Preparing to Roll Out a Patched Oracle Home

Before rolling out a patched Oracle home to your Managed Servers, you must create an Oracle home archive and distribute it to each node.

You can manually create the second Oracle home, use the OPatch utility to apply patches to it, use the `copyBinary` command to create an archive of the patched Oracle home, and then copy the archive to the nodes in your domain. See these sections for details:

The preparation process does not require you to shut down any of your Managed Servers, so there is no effect on the availability of your applications.

> **Note:**
>
> If your domain includes Oracle Fusion Middleware products other than Oracle WebLogic Server (such as Oracle SOA Suite or Oracle WebCenter), and you have patched those applications in your Oracle home, if you want to preserve currently active sessions while doing the rollout, ensure that the patched versions are compatible with ZDT patching. For example, the applied patches should have limited changes to session shape and should be backward-compatible with other Oracle Fusion Middleware products that are running in the domain.

- Creating a Second Oracle Home
- Applying Patches to the Second Oracle Home
- Creating an Archive and Distributing It to Each Node

Creating a Second Oracle Home

To manually create a patched Oracle home, you must first create a copy of your existing Oracle home by using the `copyBinary` and `pasteBinary` commands. When using these commands, you must keep in mind that the value of options specified must not contain a space. For example, on Windows, you cannot pass the following as a value to the `-javaHome` option:

```
C:\Program Files\jdk
```

> **Note:**
>
> Oracle recommends that you create and patch the second Oracle home on a nonproduction machine so that you can test the patches you apply, but this is not required. However, you must perform the following steps on the node where you will patch the new Oracle home. The Oracle home on that node must be identical to the Oracle home you are using for your production domain.

To create the second Oracle home to which you will apply patches:

1. Change to the following directory, where *`ORACLE_HOME`* is the Oracle home that you want to patch.

```
cd ORACLE_HOME/oracle_common/bin
```

2. Run the following command, where `archive` is the full path and file name of the archive file to create, and `oracle_home` is the full path to your existing Oracle home. Note that `JAVA_HOME` must be defined as the Java home that was used for your Oracle home installation:

   UNIX

   ```
   ./copyBinary.sh -javaHome $JAVA_HOME -archiveLoc archive -sourceOracleHomeLoc
   oracle_home
   ```

   Windows

   ```
   copyBinary.cmd -javaHome %JAVA_HOME% -archiveLoc archive -sourceOracleHomeLoc
   oracle_home
   ```

   For example, the following command creates the Oracle home archive `wls1221.jar` in network location `/net/oraclehomes/` using the Oracle home located at `/u01/oraclehomes/wls1221`:

   ```
   ./copyBinary.sh -javaHome $JAVA_HOME -archiveLoc /net/oraclehomes/wls1221.jar -
   sourceOracleHomeLoc /u01/oraclehomes/wls1221
   ```

3. Run the following command to create the second Oracle home, where `archive` is the full path and file name of the archive file you created, and `patch_home` is the full path to the new Oracle home to which you will apply patches. Note that `JAVA_HOME` must be defined as the Java home that was used for your original Oracle home installation:

   UNIX

   ```
   ./pasteBinary.sh -javaHome $JAVA_HOME -archiveLoc archive -targetOracleHomeLoc
   patch_home
   ```

   Windows

   ```
   pasteBinary.cmd -javaHome %JAVA_HOME% -archiveLoc archive -targetOracleHomeLoc
   patch_home
   ```

For example, the following command creates the Oracle home `wls1221_patched` in `/u01/oraclehomes/` using the archive `/net/oraclehomes/wls1221.jar`:

```
./pasteBinary.sh -javaHome $JAVA_HOME -archiveLoc /net/oraclehomes/wls1221.jar -
targetOracleHomeLoc /u01/oraclehomes/wls1221_patched
```

Applying Patches to the Second Oracle Home

To patch the second Oracle home, use the OPatch tool to apply individual patches, bundle patches, security patch updates, or patch set updates to the second, offline Oracle home. Prior to applying a particular patch or group of patches, ensure that all prerequisite patches have already been applied.

For information about how to prepare for and patch an Oracle home using OPatch, see Patching Your Environment Using OPatch in *Patching with OPatch*.

Creating an Archive and Distributing It to Each Node

After you have created the patched Oracle home, use the following steps to create an Oracle home archive and copy it to each node that will be involved in the rollout:

1. Change to the following directory, where `ORACLE_HOME` is the patched Oracle home that you created.

```
cd ORACLE_HOME/oracle_common/bin
```

2. Run the following command, where *archive* is the full path and file name of the archive file to create, and *patched_home* is the full path to the patched Oracle home you created. Note that *JAVA_HOME*must be defined as the Java home that was used for your current Oracle home installation.

   UNIX

   ```
   ./copyBinary.sh -javaHome $JAVA_HOME -archiveLoc archive -sourceOracleHomeLoc
   patched_home
   ```

   Windows

   ```
   copyBinary.cmd -javaHome %JAVA_HOME% -archiveLoc archive -sourceOracleHomeLoc
   patched_home
   ```

   For example, the following command creates the Oracle home archive `wls1221.11.jar` in network location `/net/oraclehomes/` using a patched Oracle home located at `/01/oraclehomes/wls1221_patched`:

   ```
   ./copyBinary.sh -javaHome $JAVA_HOME -archiveLoc /net/oraclehomes/wls_1221.11.jar -
   sourceOracleHomeLoc /u01/oraclehomes/wls1221_patched
   ```

3. On each node that will be included in the patching workflow, copy the archive file to the parent folder of the Oracle home that you want to replace. For example, if the archive is in network location `/net/oraclehomes/wls_1221.11.jar` and the Oracle home to be replaced is located in `/u01/oraclehomes/wls1221`:

   ```
   cp /net/oraclehomes/wls1221.11.jar /u01/oraclehomes/
   ```

   If you are copying to a large number of nodes, you can use third-party software distribution applications to perform this step.

After completing these steps, you are ready to create a workflow that includes patching your Oracle home. See Configuring and Monitoring Workflows.

Note:

If you want to also update your Java version or applications using the same patching workflow, then perform the preparation steps for those upgrades before you create the workflow.

Preparing to Upgrade to a New Java Version

Before upgrading to a new Java version, you must copy the new Java version to each node you want to include in the upgrade. Before installing the new Java version, there are certain conditions that must be met.

Preparation for upgrading to a new version of Java does not require you to shut down Managed Servers, so there will be no interruption to application availability.

To upgrade to a new version of Java:

1. Prior to installing the new Java version, ensure that Node Manager and the Managed Servers are running on all nodes on which you plan to install the new version. This prevents the Java installer from changing the existing Java home path. However, you do

not need to have the Node Manager running on the node on which the Administration Server is running.

2. On each node to be included in the upgrade, install the new Java version to the same path on each node. The full path to the new Java version must be the same on each node for the upgrade to be successful.

After copying the new Java version to each node, you are ready to create a workflow that includes upgrading to a new Java home. See Configuring and Monitoring Workflows.

Preparing to Update to New Application Versions

Before rolling out an application update, the new application version is distributed to all affected nodes depending on the staging mode you used when you staged the application. You must create a JSON file to specify the properties of applications that require an update.

This section describes how to prepare for updating to new applications using a ZDT workflow. It contains the following sections:

- The Effects of Staging Modes
- Creating an Application Update JSON File

The Effects of Staging Modes

Applications deployed across Managed Servers can be deployed using one of three staging modes: stage mode, no-stage mode, and external-stage mode. The selected mode indicates how the application will be distributed and kept up-to-date.

How you prepare for an application update workflow depends on the mode you used when you staged the application.

Staging Mode	Required Preparation and Result
Stage	Place a copy of the updated application directory on the domain's Administration Server. **Result:** The workflow will replace the original application directory on the Administration Server and WebLogic Server will copy it to each Managed Server.
No-stage	Place a copy of the updated application directory on each node that will be affected. This directory must be in the same location on each node. **Result:** The workflow will update each node in turn by replacing the existing application directory with the updated application directory, and will move the original application directory to the specified backup location.
External stage	Place a copy of the updated application directory on each node that will be affected. This directory must be in the same location on each node. **Result:** The workflow will detect that the application is an external-stage application, figure out the correct path for the stage directory for each Managed Server on the node, copy the updated application to that location, and move the original application to the specified backup location.

For detailed information about the various staging modes, see Staging Mode Descriptions and Best Practices in *Deploying Applications to Oracle WebLogic Server*.

Creating an Application Update JSON File

You can update one or more applications in your domain with a single workflow. Application updates are accomplished by creating a JSON file that, for each application, defines:

- The application name (`applicationName`)
- The path and file name for the updated application archive (`patchedLocation`)
- The path and file to which you want to back up the original application archive (`backupLocation`).

Note:

Oracle recommends that you avoid using backslash (Windows) while specifying the paths in the JSON file. This is because these paths are interpreted by Java and a backslash may trigger a different character representation.

When configuring the workflow using WLST, you must specify the name of the JSON file to use for the update.

The following example shows the structure of a JSON file that is intended to update two applications, *`MyApp`* and *`AnotherApp`*, to a new version. You can use a single JSON file to update as many applications as necessary.

```
{"applications":[
{
"applicationName":"MyApp",
"patchedLocation":"/u01/applications/MyAppv2.war",
"backupLocation": "/u01/applications/MyAppv1.war"
},
{
"applicationName":"AnotherApp",
"patchedLocation":"/u01/applications/AnotherAppv2.war",
"backupLocation": "/u01/applications/AnotherAppv1.war"
}
]}
```

After copying the updated application to all required locations and creating the JSON file, you are ready to create a workflow that includes application updates. See Configuring and Monitoring Workflows.

3

Patching an Existing WebLogic Server Installation

You can patch an existing WebLogic Server installation using either ZDT Patching or manually rolling an update of your servers.

Note:

If you are upgrading from Oracle WebLogic Server 10.3.6, you will use the Oracle OPatch instead of the BEA Smart Update (BSU) patching technology that was used in Oracle WebLogic Server 10.3.6. You will need to modify any installation and patching automation you have created for your Oracle WebLogic Server 10.3.6 environments to adapt to the new installation and patching technologies.

Use ZDT Patching only if your domain contains three or more nodes and all the servers that you want to patch are assigned to clusters.

Manually performing a rolling update of your servers results in no loss of service to your customers. Use this method to patch individual servers that are not part of a cluster or if the domain contains fewer than three nodes. See Using Zero Downtime Patching.

You can also check the list of patches that have already been applied to a WebLogic Server instance. See Obtaining a List of Applied Patches.

See the following sections about different patching methods.

- Using Zero Downtime Patching
- Obtaining a List of Applied Patches

Using Zero Downtime Patching

As of WebLogic Server 12.2.1, you can use ZDT Patching to automate the process of applying individual patches, bundle patches or patch set updates to a WebLogic Server installation.

With ZDT patching, you can use WLST to:

- Create and patch a second Oracle Home.
- Distribute the patched Oracle Home to all of your nodes.
- Configure a patching workflow to update the desired servers in your domain.

Use a patching workflow to revert patches that you have previously applied to a WebLogic Server installation using ZDT Patching.

For more details about ZDT Patching, see Introduction to Zero Downtime Patching .

Note:

OPatchAutoFMW (installed in `OPatch/auto/fmw` directory) is deprecated and is automatically removed when you update to OPatch 13.9.4.2.2 or later.

Obtaining a List of Applied Patches

Oracle WebLogic Server provides the ability to display the list of patches that have been applied to a WebLogic Server instance. The patch list can be obtained from either of the following sources:

- Using the weblogic.log.DisplayPatchInfo System Property
- Using the ServerRuntimeMBean.PatchList Attribute

When you use one of the preceding sources, the following details are provided for each applied patch:

- Associated bug number
- Patch number
- Date the patch was applied
- Brief description

Using the weblogic.log.DisplayPatchInfo System Property

The `weblogic.log.DisplayPatchInfo` system property contains a log of all patches that have been applied to a WebLogic Server instance, and can be accessed by either of the following methods:

- Specifying the `-Dweblogic.log.DisplayPatchInfo=true` JVM option in the command line that starts the server instance. As the server starts, the startup messages in `stdout` include the list of applied patches, and they are also retained in the server log file. Note that to minimize logging overhead during startup, the default value of this option is `false`.
- Running the `weblogic.version` utility. This utility can obtain the patch list regardless of whether the `-Dweblogic.log.DisplayPatchInfo=true` startup option is used, and does not require the WebLogic Server instance to be starting or running.

The following example shows running the `weblogic.version` utility. This example includes specifying the classpath of the `weblogic.jar` file corresponding to the specific server instance whose patch list is to be displayed.

```
bash-4.1$ java -classpath wlserver/server/lib/weblogic.jar weblogic.version

WebLogic Server 12.2.1.1.0 Thu Jun  2 16:21:58 PDT 2016 1784838
24907328;20845986;Mon Mar 13 14:40:42 PDT 2017;WLS PATCH SET UPDATE 12.2.1.1.170117
19795066;19149348;Mon Mar 13 14:33:28 PDT 2017;One-off
18905788;18668039;Mon Mar 13 14:32:57 PDT 2017;One-off
19632480;19278519;Mon Mar 13 14:32:26 PDT 2017;One-off
19002423;18804275;Mon Mar 13 14:31:50 PDT 2017;One-off
19030178;19234068;Mon Mar 13 14:31:22 PDT 2017;One-off
19154304;19278518;Mon Mar 13 14:30:54 PDT 2017;One-off

Use 'weblogic.version -verbose' to get subsystem information

Use 'weblogic.utils.Versions' to get version information for all modules
```

Using the ServerRuntimeMBean.PatchList Attribute

The list of patches that have been applied to a WebLogic Server instance is also available from the `ServerRuntimeMBean.PatchList` attribute. The value of this attribute is independent of the `weblogic.log.DisplayPatchInfo` system property. You can access the `ServerRuntimeMBean.PatchList` attribute using any of the following clients:

- WLST - See Example 3-1
- REST API - See Example 3-2
- JMX - See Example 3-3

Note:

To access the patch list from `ServerRuntimeMBean`, you must be an authenticated user whose identity can be mapped to the `Admin` role.

Regardless of the client that you use to obtain the patch information, each patch entry has the following format:

```
<BugNumber>;<PatchID>;<DateApplied>;<Description>
```

Example 3-1 Using WLST

The following example shows using WLST to connect to a server instance and obtain its list of applied patches:

```
wls:/offline> connect('username','password','t3://localhost:7001')
Connecting to t3://localhost:7001 with userid weblogic ...
Successfully connected to Admin Server "myserver" that belongs to domain "mydomain".

Warning: An insecure protocol was used to connect to the server.
To ensure on-the-wire security, the SSL port or Admin port should be used instead.

wls:/mydomain/serverConfig/> serverRuntime()
Location changed to serverRuntime tree.
 This is a read-only tree with ServerRuntimeMBean as the root.
For more help, use help('serverRuntime').

wls:/mydomain/serverRuntime/> print cmo.getPatchList()
array(java.lang.String,['24907328;20845986;Mon Mar 13 14:40:42 PDT 2017;WLS PATCH SET
UPDATE 12.2.1.1.170117', '19795066;19149348;Mon Mar 13 14:33:28 PDT 2017;One-off',
'18905788;18668039;Mon Mar 13 14:32:57 PDT 2017;One-off', '19632480;19278519;Mon Mar 13
14:32:26 PDT 2017;One-off', '19002423;18804275;Mon Mar 13 14:31:50 PDT 2017;One-off',
'19030178;19234068;Mon Mar 13 14:31:22 PDT 2017;One-off', '19154304;19278518;Mon Mar 13
14:30:54 PDT 2017;One-off'])
wls:/mydomain/serverRuntime/>
```

Example 3-2 Using the REST API

The following example shows using the REST API to return the patch list:

```
Request:
http://localhost:7001/management/weblogic/latest/serverRuntime?
links=none&fields=name,patchList

Response: {
    "patchList": [
        "24907328;20845986;Mon Mar 13 14:40:42 PDT 2017;WLS PATCH SET UPDATE 
12.2.1.1.170117",
        "19795066;19149348;Mon Mar 13 14:33:28 PDT 2017;One-off",
        "18905788;18668039;Mon Mar 13 14:32:57 PDT 2017;One-off",
        "19632480;19278519;Mon Mar 13 14:32:26 PDT 2017;One-off",
        "19002423;18804275;Mon Mar 13 14:31:50 PDT 2017;One-off",
        "19030178;19234068;Mon Mar 13 14:31:22 PDT 2017;One-off",
        "19154304;19278518;Mon Mar 13 14:30:54 PDT 2017;One-off"
    ],
    "name": "myserver"
}
```

Example 3-3   Using a JMX Client

Using a JMX application, you can access the applied patch list of a WebLogic Server instance by invoking the `getPatchList` method, as in the following example:

```
/**
 * @include-api for-public-api
 * Returns array of informational strings for installed patches. Each info string
 * is of the form: <bug-id>;<patch-id>;<date-applied>;<patch-description>
 * For example:
 * 24907328;20845986;Mon Mar 13 14:40:42 PDT 2017;WLS PATCH SET UPDATE 12.2.1.1.170117
 *
 * @return Array of informational strings for installed patches at a server.
 * @roleAllowed Monitor
 * @unharvestable
 */
public String[] getPatchList();
```

4

Configuring and Monitoring Workflows

Configure ZDT Patching workflows in Oracle WebLogic Server to roll out a patched Oracle home, upgrade to a new Java version, update the applications on your Managed Servers, or a combination of these tasks. Use WLST to create and monitor workflows.
This chapter describes how to configure and monitor a patching workflow that moves Managed Servers to a patched Oracle home, updates the Java version on your Managed Servers, updates the applications on your Managed Servers, or any combination of these update tasks.

> **Note:**
>
> Before initiating the update process, you must have completed all appropriate preparation steps for the type of update you are doing, as described in Preparing for Zero Downtime Patching.
>
> For Windows-based domains, before initiating a workflow to update an Oracle home, on each node, ensure that there are no locked directories or files in the Oracle home being updated, as this can prevent the Oracle home from being moved to the specified backup directory. A directory can be locked by something as simple as having a DOS command window open to that directory. A file can be locked by having it open in an application.

- Strategies for Rolling Out a Patched Oracle Home
  You must roll out the patched Oracle home to the Administration Server first before rolling it out to the targeted clusters. You can do this by either using two sequential workflows or by using a single workflow.
- Starting the Administration Server
  Before you initiate the rollout operation, it is important that you start the Administration Server using Node Manager. If there is no Node Manager configured for the Administration Server, then you can start the Administration Server by using the `startWebLogic` script.
- Using WLST to Initiate and Monitor Workflows
  WLST includes a set of ZDT Patching commands that you can use to roll out a patched Oracle home, a new Java version or a combination of both, or new application versions.

Strategies for Rolling Out a Patched Oracle Home

You must roll out the patched Oracle home to the Administration Server first before rolling it out to the targeted clusters. You can do this by either using two sequential workflows or by using a single workflow.

When you roll out a new Oracle home using WLST, you must ensure that the patched Oracle home is first rolled out to the Administration Server. There are two approaches you can take to do this:

- Use one workflow to roll out the patched Oracle home to the Administration Server, and then use a second workflow to roll out the patched Oracle home to your clusters. Oracle recommends using this approach, but it is not required.

In this scenario, using WLST, you would run either the `rolloutOracleHome` or `rolloutUpdate` command, and specify the name of the Administration Server as the target. You would then run `rolloutOracleHome` or `rolloutUpdate` again, and specify cluster targets.

- Use only one workflow to roll out the patched Oracle home to the entire domain. The workflow will automatically roll out the patched Oracle home in the Administration Server first, before rolling it out to the target clusters.

  In this scenario, using WLST, you would run either the `rolloutOracleHome` or `rolloutUpdate` command, and specify the domain name as the target.

Starting the Administration Server

Before you initiate the rollout operation, it is important that you start the Administration Server using Node Manager. If there is no Node Manager configured for the Administration Server, then you can start the Administration Server by using the `startWebLogic` script.

If the Administration Server will be included in a workflow, then you can start the Administration server using either the `startWebLogic` script or the Node Manager. The Administration Server will be automatically restarted during the rollout operation if the specified target for the rollout is a domain. However, when the rollout operation restarts the Administration Server, you might experience a brief downtime when you will not be able to connect to WLST.

To start the Administration Server before you initiate the rollout operation, you can start the Administration server in one of the following ways:

- Using the `startWebLogic` script

  If there is no Node Manager configured for the Administration Server, then you can start the Administration Server by using the `startWebLogic` script. To start the Administration Server using this script, see Starting an Administration Server with a Startup Script in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.

- Using the Node Manager

  If a Node Manager is configured for the Administration Server, then you must start the Administration Server using the Node Manager. To start the Administration Server using the Node Manager, perform the following steps:

1. If the Administration Server is running and was started using the `startWebLogic` script in the domain home, use the `stopWebLogic` command to shut it down:

   UNIX

   ```
   cd domain_home/bin
   ./stopWebLogic.sh
   ```

   Windows

   ```
   cd domain_home\bin
   stopWebLogic.cmd
   ```

2. Ensure that Node Manager is running on the host.
3. Start WLST. See Invoking WLST in *Understanding the WebLogic Scripting Tool*.
4. Use the `nmConnect` command to establish a Node Manager session. For example, use the following command to connect to the domain *`mydomain`* located in `/domains/mydomain` using SSL, where the NodeManager port is *`5556`*:

```
wls:/myserver/serverConfig> nmConnect('username', 'password', 'localhost',
'5556', 'mydomain', '/domains/mydomain','ssl')
```

5. After successfully connecting, run the `nmStart` command. For example, use the following command if the Administration Server is called `AdminServer` and the domain is located in `/domains/mydomain`:

```
nmStart('AdminServer', '/domains/mydomain')
```

See Starting the Administration Server Using Node Manager in *Administering Node Manager for Oracle WebLogic Server*.

Using WLST to Initiate and Monitor Workflows

WLST includes a set of ZDT Patching commands that you can use to roll out a patched Oracle home, a new Java version or a combination of both, or new application versions.

This section describes the WLST commands that you can use to initiate workflows to update your Managed Servers, and provides sample WLST scripts demonstrating various workflow (rollout) scenarios.

Note:

When using the WLST `rolloutOracleHome` or `rolloutUpdate` commands to initiate a rollout of a new Oracle home for a Windows-based domain, you cannot run WLST from any Oracle home that will be updated as part of the workflow. See ZDT Patching Restrictions.

Use the following WLST commands to perform automated rolling updates of your servers. You must run these commands from the Administration Server for the target domain.

- `rolloutOracleHome` — Rolls out a patched Oracle home to your Managed Servers or reverts your Managed Servers to a previous Oracle home. The patched Oracle home archive that you use in this command can be one that was created using the `copyBinary` and `pasteBinary` commands.
- `rolloutJavaHome` — Updates your Managed Servers to use a new Java version.
- `rolloutUpdate` — Updates your Managed Servers to use a patched Oracle home and a new Java version. The patched Oracle home archive that you use in this command can be one that was created using the `copyBinary` and `pasteBinary` commands.
- `rolloutApplications`—Updates specified applications that are running on your Managed Servers.

Note:

When specifying paths for Windows in rollout commands, you must use backslashes instead of forward slashes. To avoid unnecessary errors, ensure that the backslashes are escaped. (For example, `C:\\myhome\\files\\apps.json`). See Syntax for WLST Commands in *Understanding the WebLogic Scripting Tool*.

When you run one of these WLST commands, the command determines which servers need to be updated and in which order, and creates a patching workflow to update them safely. This workflow includes:

- Performing a graceful shutdown of Managed Servers one at a time. This does not include Managed Servers that are currently in ADMIN or STANDBY mode. This includes migration of singleton services if the `migrationProperties` option is included in the rollout command.
- Replacing the Oracle home directory (if applicable)
- Replacing the Java home directory (if applicable)
- Replacing application directories (if applicable)
- Restarting Node Manager on the node
- Restarting the Managed Servers on the node

Table 4-1 describes the parameters available for the WLST`rollout`commands.

Table 4-1 Arguments for WLST rollout Commands

Argument	Description
`target`	Required for all `rollout` commands. Specifies which Managed Servers will be included in the update. `target` can be one of the following: *`domain_name`* — Specify a domain name as the target if you want the Administration Server and all Managed Servers in that domain to be updated. *`clusters`* — Specify a cluster name or a comma-separated list of cluster names if you want to update all Managed Servers in the specified cluster or clusters, but not Managed Servers in other clusters. *`servers`* — Specify a server name or a comma-separated list of server names if you only want to update those Managed Servers. Note that the servers you specify must still be part of a cluster; they cannot be unclustered servers. **Note:** Typically, you should specify a server target only when updating the Administration Server. Oracle recommends that you not update individual Managed Servers in most cases as sessions may not be preserved and downtime for users may not be avoided. However, you can safely specify Managed Server targets if you have added one or more new Managed Servers and they are not at the same Java version as your other Managed Servers.
`rolloutOracleHome`	Applies only to and is required for the `rolloutOracleHome` command. Specifies the location of the Oracle home archive (JAR file) or local Oracle home directory to roll out, thereby replacing the existing Oracle home.
`backupOracleHome`	Applies only to and is required for the `rolloutOracleHome` command. Specifies the full path of the directory to which the existing Oracle home will be moved. This effectively renames the original Oracle home. For example, if your original Oracle home is `/u01/Oracle_Home`and you specify `/u01/Oracle_Home_backup`for this parameter, `/u01/Oracle_Home`will be moved (renamed) to `/u01/Oracle_Home_backup`.
`isRollback`	Optional. Applies only to the `rolloutOracleHome` and `rolloutUpdate` commands.
`javaHome`	Applies to and is required for the `rolloutJavaHome` command. Optionally, this argument may be required by the `rolloutUpdate` command. Specifies the location of the new Java home to use.

Table 4-1 (Cont.) Arguments for WLST rollout Commands

Argument	Description
`applicationProperties`	Applies to and is required for the `rolloutApplications` command. Optionally, this argument may be required by the `rolloutUpdate` command. Specifies the full path to the JSON file that defines one or more application names, application archive locations, and application backup locations.

Table 4-1 (Cont.) Arguments for WLST rollout Commands

Argument	Description
options	One or more of the following options can be included in the `rollout` commands: • `isDryRun` — If `TRUE`, the workflow operation will be evaluated but not run. The default is `FALSE`. • `isAutoRevertOnFailure` — If `TRUE`, the workflow operation should automatically revert on failure. If `FALSE`, the workflow operation will stop on a failure and you can resume or revert it. The default is `TRUE`. • `isSessionCompatible` — This option is applicable to all `rollout`commands, as it affects rollout time regardless of whether the rollout impacts session handling. The default is `FALSE`, which means that the last server to be updated on each cluster waits for all existing sessions to complete. This ensures that a compatible server is available in the cluster to handle sessions that must be served by a Managed Server that is still running on the existing version. If set to `TRUE`, this indicates that the session state in servers is 100% compatible between the existing version and the new version. Therefore, the last Managed Server in the update sequence in a cluster will shut down without waiting for all existing sessions to complete. Oracle recommends that you set this to `FALSE` unless you are absolutely sure that the session state is identical. This may cause the rollout to take longer due to the wait for session completion. **Note:** Serialization and deserialization in WebLogic Server differs slightly from Java serialization and deserialization. Therefore, additional fields on classes may result in a session being incompatible with servers on the new version, requiring that they be served by a server on the existing version. For example, a User class that adds a field such as Information will cause that session to be incompatible between versions. • `migrationProperties` — The full path to a JSON file that defines singleton service migrations to be performed during the rollout. For more information about this file and service migration, see Preparing to Migrate Singleton Services. • `shutdownTimeout` — Time (in seconds) WLST waits for a server to shut down gracefully before shutting it down forcefully. The forceful shutdown of servers may cause undesirable consequences, such as loss of session data and loss of in-flight transactions. A value of less than 1 second is ignored. If `isSessionCompatible` is set to `TRUE`, then the `shutdownTimeout` option defaults to zero, which means that WLST waits forever for the server to shut down gracefully. If `isSessionCompatible` is set to `FALSE`, then the user must specify a value for the `shutdownTimeout` option. Oracle recommends that you specify a value that gives typical applications plenty of time to complete. Because different applications have different behaviors, this value must be decided by the user. • `DelayBetweenNodes` — Use this option to specify the number of seconds to wait between the shutdown of servers on one node and the shutdown of servers on the next node in the workflow. This delay allows for: – The servers on the first node to be restarted and join the cluster – The load balancer to evenly distribute traffic – Any slow (lazy) stateful session bean clients to continue making requests before shutdown of the servers on the next node begins If not specified, this value defaults to 60 seconds. If you are not concerned about the lazy stateful session bean clients, you can include this option and set it to a lower value.

Table 4-1 (Cont.) Arguments for WLST rollout Commands

Argument	Description
	• `coherenceServiceHATarget` — Use this option to specify the High Availability (HA) Status of Coherence services on a managed Coherence server which must be met before the server is shutdown. The ZDT workflow checks and waits until all Coherence services attain the specified status. The rollout workflow can prevent cache data loss by waiting until the HA Status is met. The valid values are `none`, `machine-safe`, and `node-safe`. A value of `machine-safe` is generally preferred and ensures that a machine loss during the rollout process does not result in data loss. A value of `node-safe` ensures that loss to a single Coherence node does not result in data loss. • `coherenceServiceHAWaitTimeout` — Use this option to specify the amount of time to wait for the Coherence HA Status task in the workflow. If the HA Status is not met within the specified time, then the task times-out. The task completes and managed Coherence servers are shutdown as soon as the HA Status is met within the specified time. The default value is 60 seconds. • `extension`—The full path to the location of the extension jar file, optionally followed by a comma-separated list of script parameters specified as name-value pairs. If you specify the script parameters using this option, then these parameter values will override the values specified in the `extensionConfiguration.json` file. • `extensionProperties`—The full path to the `extensionProperties.json` file that is used to specify one or more extension jars. The `extensionProperties.json` file is typically used to specify multiple extension jars and additional extension parameters.

You can also use WLST to monitor the progress of a workflow. See Monitoring Workflow Progress.

- Rolling Out a New Oracle Home
- Updating Your Java Version
- Updating Both Oracle Home and the Java Version
- Rolling Out Updated Applications
- Reverting to the Previous Oracle Home, Java Home, or Applications
- Initiating a Rolling Restart of Servers
- Monitoring Workflow Progress
- Running, Reverting, and Resuming Stopped Workflows
- Useful WLST Commands for Workflows
- Sample WLST Script

Rolling Out a New Oracle Home

Use the `rolloutOracleHome` command if you only want to do one of the following tasks:

- Update your Administration Server to use a patched Oracle home.
- Update your entire domain (Administration Server and clustered Managed Servers) to use a patched Oracle home.
- Update clustered Managed Servers to use a patched Oracle home.

- Revert your Administration Server, clustered Managed Servers, or domain to use the previous unpatched Oracle home.

`rolloutOracleHome` has the following syntax:

```
rolloutOracleHome(target, rolloutOracleHome, backupOracleHome, [isRollback],
[options=options])
```

This command supports the `isDryRun`, `isAutoRevertOnFailure`, and `isSessionCompatible` options. You can include a comma-separated list of one or more options in this command. For information about these options, see Using WLST to Initiate and Monitor Workflows.

The following example shows how to roll out a new Oracle home to the domain *`mydomain`*. The JAR file for the patched Oracle home is located at `/net/wls/wls_patched.jar`. The original Oracle home will be moved (renamed) to `/u01/Oracle_Home_backup`. The process will not automatically revert if it fails.

```
connect('adminname', 'adminpassword', 't3://hostname:port')
domain='/domains/mydomain'
progress=rolloutOracleHome(domain, '/net/wls/wls_patched.jar',
'/u01/Oracle_Home_backup', options='isAutoRevertOnFailure=FALSE')
```

Note:

Specifying a local Oracle home directory in the `rolloutOracleHome` command is not supported when you are rolling out a new Oracle home. See ZDT Patching Restrictions.

Updating Your Java Version

Use the `rolloutJavaHome` command if you only want to do one of the following tasks:

- Update your Administration Server to use a new Java version.
- Update your entire domain (Administration Server and Managed Servers) to use a new Java version.
- Update your Managed Servers to use a new Java version.
- Revert your Administration Server, Managed Servers, or domain to use the previous Java version.

`rolloutJavaHome` has the following syntax:

```
rolloutJavaHome(target, javaHome, [options=options])
```

This command supports the `isDryRun` and `isAutoRevertOnFailure` options. You can include one or more options in a comma-separated list in this command. For information about these options, see Using WLST to Initiate and Monitor Workflows.

The following example shows how to roll out a new Java home to clusters *`Cluster1`*, *`Cluster2`*, *`Cluster3`*. The new Java home location is `/u01/jdk1.8.0_50`. The `isAutoRevertOnFailure` option is not included in this example; therefore, the workflow will automatically revert if the process fails.

```
connect('adminname', 'adminpassword', 't3://hostname:port')
clusters='Cluster1,Cluster2,Cluster3'
progress=rolloutJavaHome(clusters, '/u01/jdk1.8.0_50')
```

Updating Both Oracle Home and the Java Version

Use the `rolloutUpdate` command if you only want to do one of the following tasks:

- Update your Administration Server to use both a patched Oracle home and a new Java version.
- Update your entire domain (Administration Server and clustered Managed Servers) to use both a patched Oracle home and a new Java version.
- Update your Managed Servers to use both a patched Oracle home and a new Java version.
- Revert your Administration Server, Managed Servers, or domain to the previous Oracle home and previous Java version.

`rolloutUpdate` has the following syntax:

```
rolloutUpdate(target, rolloutOracleHome, backupOracleHome, [isRollback], [javaHome],
[options=options])
```

This command supports the `isDryRun`, `isAutoRevertOnFailure`, and `isSessionCompatible` options. You can include one or more options in a comma-separated list in this command. For information about these options, see Using WLST to Initiate and Monitor Workflows.

The following example shows how to roll out a new Oracle home and a new Java home to the Administration Server. The JAR file for the patched Oracle home is located at `/net/wls/wls_patched.jar`. The original Oracle home will be moved (renamed) to `/u01/Oracle_Home_backup`. The new Java home location is /u01/jdk1.8.0_50. The `isAutoRevertOnFailure` option is not included in this example; therefore, the workflow will automatically revert if the process fails.

```
connect('adminname', 'adminpassword', 't3://hostname:port')
server='AdminServer'
progress=rolloutUpdate(server, '/net/wls/wls_patched.jar',
'/u01/Oracle_Home_backup', '/u01/jdk1.8.0_50')
```

Rolling Out Updated Applications

Use the `rolloutApplications` command if you want to do one of the following tasks:

- Update your Managed Servers to use a new version of one or more applications.
- Revert your Managed Servers to the previous version of one or more applications.

`rolloutApplications` has the following syntax:

```
rolloutApplications(target, applicationProperties, [options=options])
```

This command supports the `isDryRun`, `isAutoRevertOnFailure`, and `isSessionCompatible` options. You can include one or more options in a comma-separated list in this command. For information about these options, see Using WLST to Initiate and Monitor Workflows.

The following example shows how to roll out the applications defined in the JSON-formatted application properties file `/u01/scratch/app_update.json` to all clusters *`Cluster1`*, *`Cluster2`*, *`Cluster3`* on a UNIX system.

```
connect('adminname', 'adminpassword', 't3://hostname:port')
clusters='Cluster1,Cluster2,Cluster3'
progress=rolloutApplications(clusters, '/u01/scratch/app_update.json')
```

Reverting to the Previous Oracle Home, Java Home, or Applications

After a successful rollout, if you want to roll back to the previous Oracle home, Java home, or application version, you must perform the following two steps to complete the rollback operation:

- Use the `rolloutUpdate` command to roll back to the previous Oracle home and Java home. However, you must keep the following restrictions in mind before you run the `rolloutUpdate` command to roll back:
  - You must specify the backed up Oracle home as the Oracle home directory to roll out. This directory should be the backup directory from the previous rollout.
  - Once you specify the backup Oracle home directory as the Oracle home directory to roll back to, you must not specify the new Java home in the command. The Java home will be automatically rolled back to the original Java home that was used in the previous Oracle home that you have specified to roll back to.
- Use the `rolloutApplications` command to rollback to the previous application version by specifying the old application archive in the json file. For more information about using this command, see Rolling Out Updated Applications

  .

The following example shows how to roll back to the previous Oracle home, Java home and applications. In this example, `myDomain` is the name of the domain to roll back to, `/pathto/unpatchedOracleHomeBackup/` is the location of the backup Oracle home directory from the previous rollout, `/pathto/unpatchedOracleHomeBackup1/` is the path of the directory to which the existing Oracle home will be moved. To enable the roll back operation, the `isRollback` parameter must be set to `true` as shown in the example:

```
rolloutUpdate('myDomain', '/pathto/unpatchedOracleHomeBackup/', '/pathto/
unpatchedOracleHomeBackup1/', 'true')
```

Initiating a Rolling Restart of Servers

Use the `rollingRestart` command if you want to do one of the following tasks:

- Initiate a rolling restart of all servers in a domain.
- Initiate a rolling restart of all servers in a specific cluster or clusters.

`rollingRestart` has the following syntax:

```
rolloutRestart(target, [options=options])
```

This command can include one or more options in a comma-separated list.

The following example shows how to perform a rolling restart of all servers in `Cluster1` and `Cluster2`.

```
connect('adminname', 'adminpassword', 't3://hostname:port')
clusters='Cluster1,Cluster2'
progress=rollingRestart(clusters)
```

Monitoring Workflow Progress

Each `rollout` command returns a `WorkFlowTaskRuntimeMBean` that you can use to poll the current status of the workflow. To monitor the progress of a rollout, use a `rollout`command in the following format:

```
progress=rollout_command
```

For example, use this command if you are rolling out a new Oracle home:

```
progress=rolloutOracleHome(DomainA, '/net/patched/wls1221p.jar',
'/net/backups/wls1221', options='isAutoRevertOnFailure=FALSE')
```

You can then use the methods of the `WorkflowTaskRuntimeMBean` to return information about the workflow. See `WorkflowTaskRuntimeMBean` in the *MBean Reference for Oracle WebLogic Server*. Here are some examples:

```
progress.getWorkflowId()
```

Returns the ID of the workflow.

```
progress.getProgressString()
'Workflow wf0011 Running: 13/36'
```

Returns a human-readable message containing information about the current workflow progress. In this example, workflow `wf0011`is currently running and has completed 13 of the 36 workflow commands.

```
progress.getStatus()
STARTED
```

Returns the current status of the workflow, which can be `STARTED`, `SUCCESS`, `RETRY`, `REVERTING`, `FAIL`, `REVERTED`, `REVERT_FAIL`, `CANCELED`, or `REVERT_CANCELED`.

The following Python script segment demonstrates one way to use the progress object to monitor a workflow and output the progress of a rollout task. Sample output is shown after the script.

```
# Print the starting information
rolloutName = progress.getName()
startTime = progress.getStartTime()
print "Started rollout task \"" + rolloutName + "\" at " + str(startTime)

# Check the state every 2 minutes
domainRuntime()
cd('RolloutService/rollout-service/ActiveWorkflows')
cd(progress.getWorkflowId())
while(get('Running')==1):
  progressString = progress.getProgressString()
  print progressString
  time.sleep(120)

# Print the ending information
endTime = progress.getEndTime()
state = progress.getState()
print "rollout \"" + rolloutName + "\" finished with state
```

Output

```
Started rollout task "Domain1Rollout" at 2014-07-22 07:29:06.528971
Running step 1 of 9
Running step 2 of 9
Running step 3 of 9
Running step 4 of 9
Running step 5 of 9
Running step 6 of 9
Running step 7 of 9
Running step 8 of 9
Running step 9 of 9
rollout "Domain1Rollout" finished with state "SUCCESS" at
2014-07-22 07:47:15.538299
```

Running, Reverting, and Resuming Stopped Workflows

A workflow can stop in either the running or reverting direction for the following reasons:

- The workflow failed while running, with the `isAutoRevertOnFailure` option set to `FALSE`.
- The workflow was manually canceled.
- An unrecoverable error occurred during a revert operation.

When a workflow is stopped, you can resolve any errors manually. You can then set the workflow to continue to run or revert by using the following methods on the `RolloutServiceRuntimeMBean`:

Method	Description
`executeWorkflow(WorkflowTaskRuntimeMBean)`	Takes a progress object that is eligible to be resumed and resumes it in the execute direction. If the last successful operation on the workflow was an execute, then the execute will resume with the next execute step. If the last successful operation on the workflow was a revert, then the execute will resume by running that revert step.
`revertWorkflow(WorkflowTaskRuntimeMBean)`	Takes a progress object that is eligible to be resumed and resumes it in the revert direction. If the last successful operation on the workflow was an execute, then the revert will resume with that step. If the last successful operation on the workflow was a revert, then the revert will resume by reverting the next step in the revert sequence.
`canResume(WorkflowTaskRuntimeMBean)`	Returns `true` if the workflow stopped before it was completed and is not currently running in either direction. A workflow in this state is eligible to be resumed in either the execute or revert direction.

Useful WLST Commands for Workflows

This section describes several WLST commands that you may find useful.

- **To get a list of completed workflows:**

  ```
  wls:/domain_name/domainRuntime/RolloutService/rollout-service> completeWfs=
  cmo.getCompleteWorkflows()
  ```

- **To get a list of active workflows:**

  ```
  wls:/domain_name/domainRuntime/RolloutService/rollout-service> activeWfs =
  cmo.getActiveWorkflows()
  ```

- **To look up a workflow by ID and retrieve its status:**

  ```
  wls:/domain_name/domainRuntime/RolloutService/rollout-service>
   progress=cmo.getWorkflowTask('workflow_id')
  wls:/Domain1221/domainRuntime/RolloutService/rollout-service> progress.getStatus()
  ```

- **To cancel a running workflow:**

  ```
  wls:/domain_name/domainRuntime/RolloutService/rollout-service>
   progress=cmo.getWorkflowTask('workflow_id')
  wls:/domain_name/domainRuntime/RolloutService/rollout-service> progress.cancel()
  ```

- **To delete a completed workflow:**

  ```
  wls:/domain_name/domainRuntime/RolloutService/rollout-service>
  cmo.deleteWorkflow('workflow_id')
  ```

Sample WLST Script

This section contains a sample WLST script that illustrates how to perform a rolling restart of all servers in a cluster called `cluster1` with single service migration. In this script, the following arguments are defined:

- `username` — The WebLogic Server administrator user name.
- `password` — The WebLogic Server administrator password.
- `adminURL` — The host name and port number of the domain's Administration Server.
- `target` — The target or targets for the operation. See Table 4-1.
- `options` — The rollout option or options for the operation. See Table 4-1.

The following example shows a sample WLST script for a rollout operation.

```
import sys, socket
import os
import time
from java.util import Date
from java.text import SimpleDateFormat

argUsername = sys.argv[1]
argPassword = sys.argv[2]
argAdminURL = sys.argv[3]
```

```
argTarget = sys.argv[4]
argOptions = sys.argv[5]

try:
   connect(argUsername, argPassword, argAdminURL)
   progress = rollingRestart(argTarget, argOptions)
   lastProgressString = ""

   progressString=progress.getProgressString()
   # for testing progressString="12 / 12"
   steps=progressString.split('/')


   while not (steps[0].strip() == steps[1].strip()):
     if not (progressString == lastProgressString):
       print "Completed step " + steps[0].strip() + " of " + steps[1].strip()
       lastProgressString = progressString

     java.lang.Thread.sleep(1000)

     progressString=progress.getProgressString()
     steps=progressString.split('/')
     if(len(steps) == 1):
       print steps[0]
       break;

   if(len(steps) == 2):
     print "Completed step " + steps[0].strip() + " of " + steps[1].strip()

   t = Date()
   endTime=SimpleDateFormat("hh:mm:ss").format(t)

   print ""
   print "RolloutDirectory task finished at " + endTime
   print ""

   state = progress.getStatus()
   error = progress.getError()

   stateString = '%s' % state
   if stateString != 'SUCCESS':
     #msg = 'State is %s and error is: %s' % (state,error)
     msg = "State is: " + state
     raise(msg)
   elif error is not None:
     msg = "Error not null for state: " + state
     print msg
     #raise("Error not null for state: %s and error is: %s" + (state,error))
     raise(error)
except Exception, e:
  e.printStackTrace()
  dumpStack()
  raise("Rollout failed")

exit()
```

To run this script, save it in a Python (`.py`) file and then enter commands similar to this. If you are running WLST on Windows, see ZDT Patching Restrictions, for important information about using WLST on Windows.

```
$ORACLE_HOME/oracle_common/common/bin/wlst.sh
/u01/scripts/rollout/RollingRestart.py username password
t3://hostname:port cluster1 "migrationProperties=/u01/json/mig.txt"
```

5

Modifying Workflows Using Custom Hooks

Modify the existing ZDT Patching workflow in Oracle WebLogic Server by adding custom logic specific to your business at predefined points called extension points.

- About Extension Points
- The Patching Workflow Process for Custom Hooks
- Specifying Extensions to Modify the Workflow

About Extension Points

Extension points are placeholders in the ZDT Patching workflow where you can insert custom logic. ZDT Patching provides extension points and predefined environment variables for rollouts.

The ZDT custom hooks feature identifies certain points in a patching workflow where additional commands can be run to customize its behavior. These points are referred to as extension points. You can customize the behavior of a workflow by inserting collections of resources, called extensions, at each predefined extension point.

Table 5-1 lists the available extension points for workflows along with their descriptions and use cases.

Table 5-1 Extension Points Available for Workflows

Name	Description	Use Cases
ep_OnlineBeforeUpdate	Use this extension point at the initial stage of the workflow before the patching operation starts on each node. This is typically the point where prerequisite checks can be performed.	• Pre-upgrade quiesce to disable or pause external domains that are fed into the cluster. • Run any SQL script that may be needed to prepare for an application update.
ep_EachNode	Use this extension point when the workflow needs to perform any additional operation across each node.	• Add checks to ensure that there is enough disk space on all the nodes for the rollout of Oracle home. • Ensure that any new shared file system artifacts are accessible on each node.
ep_OfflineBeforeUpdate	Use this extension point at the stage of the workflow when all servers are shut down, just before the Oracle home or Java home update starts.	• Back up files or directories.

Table 5-1 (Cont.) Extension Points Available for Workflows

Name	Description	Use Cases
ep_OfflineAfterUpdate	Use this extension point to perform any custom operation after Oracle home or Java home have been patched and before the servers start.	• Validate the versions of software components included in the rollout. • Modify any Java properties in the Java home.
ep_OnlineAfterServerStart	Use this extension point to perform any custom operation after the update has completed on each node and the server has restarted.	Perform server and application-level administrative tasks, such as: • Check JDBC, JTA or JMS subsystem. • Deploy or redeploy any additional application at this point. • Directing administrative requests to applications to ensure that the applications behave as expected.
ep_OnlineAfterUpdate	Use this extension point to perform any additional operation after the servers have restarted, and the application is continuing servicing requests.	• Perform any basic checks to ensure that affected applications are functional and accessible.
ep_RolloutSuccess	Use this extension point to define any custom logic, such as sending notifications, after the patching is successful.	• Send out an e-mail to the administrator to notify the status of the upgrade.

This feature also provides certain predefined environment variables that can be passed at the extension points. Some predefined environment variables are available for use with online extension points, while others can be used with offline extension points. Online extension points can be run when the server is running, whereas offline extension points are available for use when the server is shut down. Both offline and online extension points can be run either on the remote node or on the local node. Table 5-2 provides a list of all the environment variables that are available for use with online and offline extension points.

Table 5-2 Predefined Environment Variables for Extension Points

Variable Name	Description	Available for Use with Offline or Online Extension points
javaHome	The location of the existing Java home	Offline
newJavaHome	The location of the new Java home to use	Offline
mwHome	The location of the Middleware home	Offline
domainDir	The location of the domain directory	Offline

Table 5-2 (Cont.) Predefined Environment Variables for Extension Points

Variable Name	Description	Available for Use with Offline or Online Extension points
domainTmp	The location of the directory under the domain home where temporary files may be stored	Offline
patched	The location of patched Oracle home	Offline
backupDir	The location where the existing Oracle home will be moved	Offline
isRevert	Controls run or revert operations in scripts	Offline
currentNodeName	The full name of the node currently being updated. This variable is not applicable to the `ep_OnlineBeforUpdate` or `ep_RolloutSuccess` extension points.	Online
currentServerNames	A comma-separated list of names of servers on the targeted node. This variable is not applicable to `ep_OnlineBeforeUpdate` or `ep_RolloutSuccess` extension points.	Online
applicationInfo	The application name, application location, and application backup, separated by commas for each application. Separate multiple applications by colon: `<appName>,<appLoc>,<appBackUp>:<appName2>,<appLoc2>,<appBackUp2>` For example, "`scrabble,/pathTo/scrabblev2,/pathTo/scrabbleV1Backup:cart,/pathTo/cartV3,/pathTo/cartV2Backup`	Online

The Patching Workflow Process for Custom Hooks

You can customize operations in the workflow that is run either on the Administration server node or on a remote node.

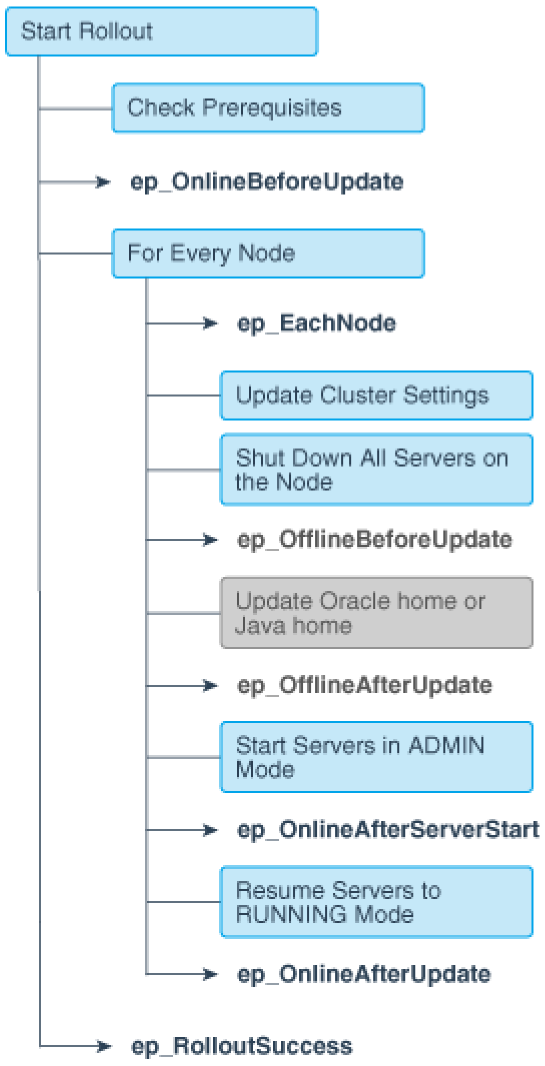
When the workflow process reaches a user hook, the user specified extension at that extension point is run. Any script that exits with a code of zero is considered to have completed successfully, whereas a script that returns a non-zero exit code is considered failed. If no errors occur, the processing resumes. If an error occurs during the running of an extension, then the workflow is rolled back to its previous state. Note that the scripts do not have in-built retry or resume methods. A script is attempted once, and if it fails, the workflow does not retry it. Therefore, if your script performs an operation that needs to be retried, then you must write the retry logic in the script.

During the workflow, the output generated by offline scripts to STDOUT or STDERR is propagated to the Administration Server log file. Similarly, output generated by a script that is run locally is also written to the Administration Server log file. This includes error output or non-error output.

Figure 5-1 illustrates the typical scenarios for workflows, and how they include different extension points.

Figure 5-1 Patching Workflow with Extension Points

This figure shows the extension points available in a typical workflow for updating Oracle home or Java home.



Specifying Extensions to Modify the Workflow

The custom hooks feature provides several ways to introduce extensions in the workflow. You can specify the extensions in either of the two JSON files, `extensionConfiguration.json` or `extensionProperties.json`, or pass them directly as options in the rollout commands.

Regardless of how you pass the extension parameters, these parameters ultimately map to script parameters that are translated into environment variables.

This flexibility lets you override or customize parameters at different levels. When you use more than one way of specifying extension parameters, then the following is the order in which script parameters are overridden:

- Extension parameters specified in the `extensionConfiguration.json` file.
- Extension parameters specified in the extension properties JSON file to override the parameters set in the `extensionConfiguration.json` file.
- Extension parameters specified as options in the WLST rollout commands to override the parameters specified in the two JSON files. You can use the same extension JAR in different environments by customizing only the options in the rollout commands for each workflow.

The following sections provide more information about using these methods to specify extensions.

Creating a JSON Configuration File

The `extensionConfiguration.json` file is a JSON format file that contains an array of extension definitions. Each extension definition must specify the following:

- The name of the predefined extension point where the extension is inserted in the workflow.
- The fully qualified name of the class file to run at that extension point.

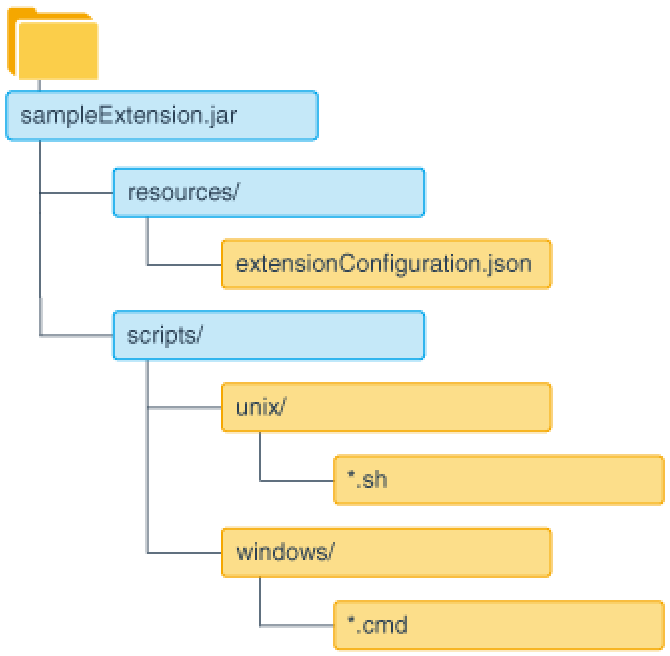
Optionally, any additional parameters used by the extension. The additional extension parameters must be declared in the JSON format. The specified class file can use one of the standard extensions supplied by WebLogic Server. The following sample `extensionConfiguration.json` file shows how to define extensions.

```
{"extensions":[
{
"extensionPoint":"ep_OnlineBeforeUpdate",
"extensionClass":"weblogic.management.patching.extensions.ScriptExecutorExtens
ion",
"extensionParameters":{"scriptName":"checkJar.sh","jarPath":"/tmp/
extension.jar"}
},
{
"extensionPoint":"ep_EachNode",
"extensionClass":"weblogic.management.patching.extensions.ScriptExecutorExtens
ion",
"extensionParameters":{"scriptName":"checkDiskSpace.sh"}
},
{
"extensionPoint":"ep_OnlineAfterUpdate",
"extensionClass":"weblogic.management.patching.extensions.ScriptExecutorExtens
ion",
"extensionParameters":{"scriptName":"checkApps.sh","appUrls":"http://
localhost:8004/Coke/Simple_stage/handle,http://localhost:8006/Coke/
Simple_stage/handle"}
},
{
"extensionPoint":"ep_RolloutSuccess",
"extensionClass":"weblogic.management.patching.extensions.ScriptExecutorExtens
ion",
"extensionParameters":{"scriptName":"emailSuccess.sh"}
}
]}
```

After you create the `extensionConfiguration.json` file, you must place it along with other related native scripts in a JAR file, such as, *`sampleExtension`*`.jar`. The JAR file that you create must have a directory structure that adheres to Oracle standards. If more than one extension is specified at a single extension point, then the extensions are run in the order in which they appear in the `extensionConfiguration.json` file. Each JAR file should contain only one `extensionConfiguration.json` file. Figure 5-2 shows the structure of an extension JAR file.

During the rollout, the scripts are extracted in the `patching` directory under *`DOMAIN_HOME`*`/bin`.

Figure 5-2 Extension Jar File Structure



Creating a JSON Properties File

Alternatively, you can specify the extension information in another JSON file, such as, `extensionProperties.json` file. You can use this file when you need to pass multiple extensions in a workflow and when these extensions are placed in multiple JAR files. Note that the JSON properties file is different from the `extensionConfiguration.json` file; the `extensionConfiguration.json` file is specific to the scripts within its own JAR file, whereas the JSON properties file gives you a convenient way to include multiple extension JAR files in the workflow. Each JSON properties file includes the path to one or more JAR files that contain the extension configuration information and optionally includes any additional parameters.

The following snippet shows the format of a sample `extensionProperties.json` file.

```
{"extensionProperties":[
{
"extensionJar":"/pathTo/extension.jar",
"extensionParameters":{"scriptName":"updateProperties.sh", "appURL":"http://
localhost:7005/context?param1=val1&param2=val2,http://localhost:7006/context2?
param1=val1&param2=val2"}
}
]}
```

Including Options in the WLST Rollout Commands

You can pass extension parameters in either of the two JSON files or pass them directly to the WLST rollout commands using the `extension` or `extensionProperties` options. For more information about how to use these options to specify extension parameters, see the arguments for WLST rollout commands in Using WLST to Initiate and Monitor Workflows.

Note:

When you create JSON files to include your extensions and place them in extensions jars, be sure to meet the following conditions:

- Place extension jars on all remote nodes before the rollout.
- Specify the path to the extension jar that contains the extension parameters to roll out. The same path must exist on all nodes.
- On a Windows system, avoid using the backslash character when you specify the paths in the JSON file.
- Do not include commas in the values of the script parameters in the JSON files.