Oracle Fusion Middleware

Administering the WebLogic Persistent Store

15c (15.1.1.0.0)
G31572-01
October 2025

Oracle Fusion Middleware Administering the WebLogic Persistent Store, 15c (15.1.1.0.0)

G31572-01

Primary Author: Oracle Corporation

Contents

Preface

This document describes how to configure and monitor WebLogic Persistent Store.

Audience

This document is a resource for system administrators and operators responsible for implementing a WebLogic Server installation. This document is relevant to all phases of a software project, from development through test and production phases.

It is assumed that the reader is familiar with Jakarta EE and Web technologies, object-oriented programming techniques, and the Java programming language.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

Related Documentation

- *Understanding Domain Configuration for Oracle WebLogic Server*
- *Oracle WebLogic Remote Console Online Help*

New WebLogic Server Features

For a comprehensive listing of the new WebLogic Server features introduced in this release, see *What's New in 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

The WebLogic Persistent Store

This chapter explains how to configure and monitor the WebLogic Server persistent store, which provides a built-in, high-performance storage solution for WebLogic Server subsystems and services that require persistence. It also describes how to configure high availability for JMS service artifacts that use persistent stores.

What is a Persistent Store

The persistent store provides a built-in, high-performance storage solution for WebLogic Server subsystems and services that require persistence.

For example, it can store persistent JMS messages or temporarily store messages sent using the Store-and-Forward feature. The persistent store supports persistence to a file-based store or to a JDBC-accessible store in a database.

Table 1-1 defines many of the WebLogic services and subsystems that can create connections to the persistent store. Each subsystem that uses the persistent store specifies a unique connection ID that identifies that subsystem.

Table 1-1 Persistent Store Users

Subsystem/Service	What It Stores	More Information
Diagnostic Service	Log records, data events, and harvested metrics.	Understanding WLDF Configuration in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*
JMS Messages	Persistent messages and durable subscribers.	Understanding the Messaging Models in *Developing JMS Applications for Oracle WebLogic Server*
JMS Paging Store	One per JMS server. Paged persistent and non-persistent messages.	Main Steps for Configuring Basic JMS System Resources in *Administering JMS Resources for Oracle WebLogic Server*.
JTA Transaction Log (TLOG)	Information about committed transactions coordinated by the server that may not have been completed. TLOGs can be stored in the default persistent store or a JDBC TLOG store.	• Managing Transactions in *Developing JTA Applications for Oracle WebLogic Server*. • Using a JDBC TLog Store
Path Service	The mapping of a group of messages to a messaging resource.	Using the WebLogic Path Service in *Administering JMS Resources for Oracle WebLogic Server*
Store-and-Forward (SAF) Service Agents	Messages for a sending SAF agent for retransmission to a receiving SAF agent	Understanding the Store-and-Forward Service in *Administering the Store-and-Forward Service for Oracle WebLogic Server*.
EJB Timer Services	EJB Timer objects.	Understanding Enterprise JavaBeans in *Developing Jakarta Enterprise Beans Using Deployment Descriptors*

See Monitoring Store Connections.

Features of the Persistent Store

This section describes the key features of the persistent store.

The key features of the persistent store include:

- Default file store for each server instance that requires no configuration.
- The Default and custom stores are shareable by multiple subsystems, as long as they are all targeted to the same server instance, cluster, or migratable target.
- When configured, a JDBC TLOG store which contains information about committed transactions coordinated by the server that may not have been completed. You can select to persist TLOG information either in the default store or the JDBC TLOG store, depending on your application needs. See Using a JDBC TLog Store.
- High-performance throughput and transactional support.
- Modifiable parameters that let you create customized file stores and JDBC stores.
- Monitoring capabilities for persistent store statistics and open store connections.
- In a clustered environment, the JDBC TLOG store and customized stores can be migrated from an unhealthy server to a backup server, either on the whole-server level or on the service level.
- When targeted to a cluster, the high availability parameters of the persistent store control the distribution and high availability behavior of JMS services. It also eliminates the need to configure Migratable Targets. See Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*.

High-Performance Throughput and Transactional Support

Throughput is the main performance goal of the persistent store. Multiple subsystems can share the same default or custom store, as long as they are all targeted to the same server instance, cluster, or migratable target.

> **Note**
>
> - The JDBC TLOG store is only used to persist information about committed transactions coordinated by the server that may not have been completed. It can not be shared by other subsystems.
> - The JDBC TLOG store does not allow HA configuration settings.

This is a performance advantage because the persistent store is treated as a single resource by the transaction manager for a particular transaction, even if the transaction involves multiple services that use the same store. For example, if the TLOG, JMS and EJB timers share a file store, and a JMS message and an EJB timer are created in a single transaction, the transaction will be one-phase and incur a single resource write, rather than two-phase, which incurs four resource writes (two on each resource), plus a transaction entry write (on the transaction log).

Both a file store and a JDBC store can survive a process crash or hardware power failure without losing any committed updates. Uncommitted updates may be retained or lost, but in no case will a transaction be left partially complete after a crash.

Comparing File Stores and JDBC-accessible Stores

This section describes the similarities and differences between file stores and JDBC-accessible stores.

The following are some similarities and differences between file stores and JDBC-accessible stores:

- The default persistent store can only be a file store. Therefore, a JDBC store cannot be used as a default persistent store.
- Both have the same transaction semantics and guarantees. As with JDBC store writes, file store writes are guaranteed to be persisted to disk and are not simply left in an intermediate (that is, unsafe) cache.
- Both have the same application interface (no difference in application code).
- All things being equal, file stores generally offer better throughput than a JDBC store.

> **Note**
>
> If a database is running on high-end hardware with very fast disks, and WebLogic Server is running on slower hardware or with slower disks, then you may get better performance from the JDBC store.

- File stores are generally easier to configure and administer, and do not require that WebLogic subsystems depend on any external component.
- File stores generate no network traffic; whereas, JDBC stores generate network traffic if the database is on a different machine from WebLogic Server.
- JDBC stores may make it easier to handle failure recovery since the JDBC interface can access the database from any machine on the same network. With the file store, the disk must be shared or migrated.
- **Dynamic Scalability**: When custom *logical* persistent stores are configured and targeted to a cluster, by default, the system automatically creates one *physical* instance on each of the cluster member, and the instance is named uniquely for monitoring purposes. This allows the store and related JMS artifacts to dynamically scale without the need for individually configuring them on each cluster member. This behavior can be changed such that the system only creates one *physical* instance and make it high available in the cluster. See Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*.

High Availability For Persistent Stores

This section describes high availability options offered by the WebLogic Server.

Whole Server Migration

A persistent file-based store (default, or custom) can be migrated along with its parent server as part of the "whole server-level" migration feature, which provides both automatic and

manual migration at the server level, rather than on the service level. See Whole Server Migration in *Administering Clusters for Oracle WebLogic Server*. However, file-based stores must be configured on a shared disk that is available to all servers in the cluster.

Automatic Service Migration

File-based stores and JDBC-accessible stores can also be migrated as part of a "service-level" migration for JMS-related services, such as JMS servers, SAF agents, and the path service, which rely on stores to maintain data. WebLogic Server supports automatic service migration in two ways:

- By using simplified JMS cluster configuration: This enables the automatic service migration for both store and all the JMS service artifacts that reference the store. The configuration settings will take effect whenever the store is targeted to a cluster. This model offers enhanced HA capabilities such as automatic failback, dynamic load balancing, and failover. See Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*.
- By using Migratable Target configuration: In this model, a migratable target serves as a grouping mechanism for related JMS services, and the entire group is hosted on only one physical server in a cluster.

> **Note**
>
> For automatic service migration, use simplified JMS cluster configuration instead of the legacy migratable target model.

In both these models, the related hosted services can be automatically migrated from the current unhealthy hosting server to a healthy active server with the help of the Health Monitoring subsystem. In a cluster targeted Store case, when any store instance migrates, all the associated JMS service instances that are referencing that Store instances are also migrated.

In this release, Service-level migration is controlled by targeting the Store to the same cluster as the associated JMS service artifacts, with appropriate high availability parameter settings on the Store. See Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*. This type of migration is supported in all the cluster types (configured, dynamic, and mixed) and eliminates the need for Migratable Target configuration. This option also supports automatic failback as well as it controls the service migration of the JMS artifact.As in the previous releases, you can still enable Service-level migration by targeting related JMS services to a Migratable Target, which serves as a grouping of JMS-related services and which is hosted on only one physical server in a cluster. In Migratable Target based configuration, the JMS services hosted by a migratable target can also be manually migrated on demand as part of regularly scheduled server maintenance.

In both the cases, JMS services can be automatically migrated from the current unhealthy hosting server to a healthy active server with the help of the Health Monitoring subsystem. When the migration takes place, all pinned services associated with the Store and are hosted by that Server are also migrated.

See Service Migration in Administering Clusters for Oracle WebLogic Server.

In the cluster or migratable target based model, JMS-related services cannot use the default file store, so you must configure a custom file store or JDBC store and target it to the same migratable target as the JMS server, SAF agent, or path service associated with the store.

For best practices, see Additional Requirement for High Availability File Stores.

Service Restart In Place

Service Restart In Place provides options to automatically recover a failed custom store and its dependent services on their original running WebLogic Server. For information about Service Restart In Place for other store types and messaging bridges, see the Service Restart In Place in Combination with Migration and Additional Notes sections below.

When Restart In Place is not configured and in effect, WebLogic Server marks failed custom stores and their dependent JMS services as unhealthy and shuts them down. For example, this can happen when a file store gets an error from a file system or when a JDBC store cannot access its database. Messages persisted prior to a store shutdown are unavailable for consumption until either the store is restarted or is migrated to another server within the same cluster.

The way to enable Service Restart In Place on a custom store varies based on the store target.

Custom Store Target	Service Restart In Place Option
Standalone server or cluster	Option 1: Explicitly configure the store `Restart In Place` setting to true. Option 2: Set the store `Migration Policy` to `Always` or `On-Failure`. This causes the `Restart In Place` setting to default to `true`. With either option, you can fine-tune Restart In Place behavior by changing `Seconds Between Restarts` (default 30) and `Number Of Restart Attempts` (default 6) in the store configuration.
Migratable target	Enable Restart In Place on the migratable target. You can fine-tune Restart In Place behavior by changing `Seconds Between Restarts` (default 30) and `Number Of RestartAttempts` (default 6) in the migratable target configuration. See In-Place Restarting of Failed Migratable Services in *Administering Clusters for Oracle WebLogic Server*.
Managed Server instance within a cluster	It is not possible to enable Restart In Place on stores that are directly targeted to a server within a cluster. Oracle recommends targeting stores and their dependent services to the cluster or to a migratable target instead.

Service Restart In Place in Combination with Migration

Service Restart In Place can be configured independently of whole server migration or service migration. When Restart In Place and migration are both configured, they work as follows:

Restart In Place and Service Migration

If Restart In Place is enabled, if the store's original host JVM is still running, and if a failed store is configured to migrate from one server to another within a WebLogic cluster, then the system tries to restart the store on its original host JVM before it tries a migration. See Service Migration in *Administering Clusters for Oracle WebLogic Server*.

Restart In Place and Whole Server Migration

If a globally-scoped store is targeted to a standalone server, is targeted to a server within a cluster, or is targeted to a cluster and has a MigrationPolicy of Off, then the store places its host WebLogic Server instance in a failed health state after all of its restart attempts fail. The failed WebLogic Server health state allows the optional Whole Server Migration framework to detect the problem and attempt to either restart the WebLogic Server JVM or to migrate the JVM to another server. See Whole Server Migration in *Administering Clusters for Oracle WebLogic Server*.

Additional Notes

- Service Restart In Place is not applicable to WebLogic default stores, Transaction Log Store, or messaging bridges.
  - Failed default stores cause a server to enter a Failed health state, and require a Whole Server Migration or Whole Server restart to recover. Oracle recommends that you configure services to persist critical information in custom stores instead of default stores.
  - Messaging Bridges ignore Restart In Place settings. Instead, they automatically handle failures by periodically retrying when they fail to connect to their source or target destinations.
- Custom JDBC Stores have an additional internal retry mechanism that takes effect first before they shutdown and requires the store and its dependent services to restart. This functionality is helpful for silent recovery from brief database outages. See Configuring JDBC Store Reconnect Retry.

High Availability Storage Solutions

If you have applications that need access to persistent stores that reside on remote machines after the migration of a JMS server or JTA transaction log, then you should implement one of the following highly-available storage solutions:

- File-based stores (default or custom)—Implement a hardware solution, such as a dual-ported SCSI disk or Storage Area Network (SAN) to make a file store available from shareable disks or remote machines.

  > **Note**
  >
  > - Persistent file stores that may migrate to a different JVM or machine must be explicitly configured to reference a shared directory. See Additional Requirement for High Availability File Stores.
  > - If a file store is disconnected and re-connected again, its host server instance must be rebooted to successfully continue sending/receiving persistent JMS messages. For example, if for some reason the file system containing a file store is unmounted and then remounted, attempts to send persistent JMS messages will generate JMS exceptions until the host server is rebooted.

- JDBC-accessible stores—Configure a JDBC store or JDBC TLOG store and use JDBC to access this store, which can be on yet another server. Applications can then take advantage of any high-availability or failover solutions offered by your database vendor. In addition, JDBC stores support GridLink data sources and multi data sources, which

provide failover between nodes of a highly available database system, such as Oracle Real Application Clusters (Oracle RAC). For more information, see:

  - Configuring JDBC Multi Data Sources in *Administering JDBC Data Sources for Oracle WebLogic Server*
  - Using GridLink Data Sources in *Administering JDBC Data Sources for Oracle WebLogic Server*

- Any persistent store—Use high-availability clustering software which provides an integrated, out-of-the-box solution for WebLogic Server-based applications.

Limitations and Considerations of the Persistent Store

This section describes the limitations applicable to the persistent store.

- A persistent file store should not be opened simultaneously by two server instances; otherwise, there is no guarantee that the data in the file will not be corrupted. If possible, the persistent store will attempt to return an error in this case, but it will not be possible to detect this condition in every case. It is the responsibility of the administrator to ensure that the persistent store is being used in an environment in which multiple servers will not try to access the same store at the same time. (Two file stores are considered the "same store" if they have the same name and the same directory.)
- Two JDBC stores must not share the same database table, because this will result in data corruption. A JDBC store will normally prevent this from happening by detecting if a backing table has already been opened by another instance, but it is not possible to detect this condition in every case. It is the responsibility of the administrator to ensure that the persistent store is being used in an environment in which multiple servers will not try to access the same store at the same time. (Two JDBC stores can reference the same table if they have the same table name prefix and database schema.)
- A persistent store may not survive arbitrary corruption. If the disk file is overwritten with arbitrary data, then the results are undefined. The store may return inconsistent data in this case, or even fail to recover at all.
- A file store may return exceptions when its disk is full. However, it will resume normal operation by no longer throwing an exception when disk space has been made available. Also, the data in the persistent store must remain intact as described in the previous points.
- When using MySQL as the backing database for a JDBC store, Oracle recommends using the `InnoDB` engine because it provides safe writes. If the `MyISAM` engine is used, data may be lost in some cases.

Additional Requirement for High Availability File Stores

Custom and default file stores that are configured for high availability via service migration or whole server migration must explicitly configure a directory on a central location on a shared disk.

This ensures that the same directory and files are available to all servers and machines that may host a store, and is required to ensure that a store can recover its data after it migrates.

This applies to default and custom file store locations, it does not apply to cache or page file directories as the latter do not need to be highly available and can and should be located on a local drive for performance reasons.

See File Locations.

See Migratable Target and Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*.

File Locations

Persistent stores create a number of files in the file system for different purposes. Among them are file store data files, file store cache files (for file stores with a `DirectWriteWithCache` synchronous write policy), and JMS server and SAF agent paging files.

Table 1-2 describes the location of various files used by the file store system at the domain level.

Table 1-2 File Locations

Store Type	Store Path Not Configured	Relative Store Path	Absolute Store Path	File Name
`default`	`<domainRoot>/servers/<serverName>/data/store/default`	`<domainRoot>/<relPath>`	`<absPath>`	`_WLS_<serverName>NNNNNN.DAT`
`custom file`	`<domainRoot>/servers/<serverName>/data/store/<storeName>`	`<domainRoot>/<relPath>`	`<absPath>`	`<storeName>NNNNNN.DAT`
`cache`	`${java.io.tmpdir}/WLStoreCache/${domainName}/<storeUuid>`	`<domainRoot>/<relPath>`	`<absPath>`	`<storeName>NNNNNN.CACHE`
`paging`	`<domainRoot>/servers/<serverName>/tmp`	`<domainRoot>/<relPath>`	`<absPath>`	`<jmsServerName>NNNNNN.TMP` `<safAgentName>NNNNNN.TMP`

Table 1-3 shows how each of the prior store types configure their directory location.

Table 1-3 Store Type Directory Configuration

Store Type	Directory Configuration
`default`	The directory configured on a WebLogic Server default store. See Using the Default Persistent Store.
`custom file`	The directory configured on a custom file store. See Using Custom File Stores.
`cache`	The cache directory configured on a custom or default file store that has a `DirectWriteWithCache` synchronous write policy. See Tuning the WebLogic Persistent Store in *Tuning Performance of Oracle WebLogic Server*.

Table 1-3 (Cont.) Store Type Directory Configuration

Store Type	Directory Configuration
`paging`	The paging directory configured on a SAF agent or JMS server. See Paging Out Messages To Free Up Memory in *Tuning Performance of Oracle WebLogic Server*.

2

Using the Default Persistent Store

This chapter explains how to configure and monitor the WebLogic Server persistent store, which provides a built-in, high-performance storage solution for WebLogic Server subsystems and services that require persistence.

Using the Default Persistent Store

Each server instance, including the Administration Server, has a default persistent store that requires no configuration. The default store is a file-based store that maintains its data in a group of files in a server instance `data\store\default` directory. A directory for the default store is automatically created if one does not already exist.

This default store is available to subsystems that do not require explicit selection of a particular store and function best by using the system's default storage mechanism. For example, a JMS Server with no persistent store configured will use the default store for its Managed Server and will support persistent messaging.

The default store can be configured by directly manipulating DefaultFileStoreMBean parameters. If this MBean is not defined in the domain configuration file, then the configuration subsystem ensures that the `DefaultFileStoreMBean` always exists with the default values.

Default Store Location

The default store maintains its data in a `data\store\default` directory inside the *`servername`* subdirectory of a domain's root directory.

For example, if no directory name is specified for the default file store, it defaults to:

`ORACLE_HOME``\user_projects\domains\`*`domain-name`*`\servers\`*`server-name`*`\data\store\default`

where *`domainname`* is the root directory of your domain, typically `c:\oracle\user_projects\domains\domainname`, which is parallel to the directory in which WebLogic Server program files are stored, typically `c:\oracle\wlserver_12.1`.

Example of a Default File Store

This section provides an example of how a default file store may look in a domain's configuration file, with the default directory location and Synchronous Write Policy settings overridden.

```
<server
  <name>myserver</name>
  <default-file-store>
  <directory>C:/store</directory>
  </default-file-store>
</server>
```

3

Using Custom Persistent Stores

This chapter explains how to configure and monitor the WebLogic Server persistent store, which provides a built-in, high-performance storage solution for WebLogic Server subsystems and services that require persistence.

What are Custom File Stores and JDBC Stores

In addition to using the default file store, you can also configure a file store or JDBC store to suit your specific needs. A custom file store, like the default file store, maintains its data in a group of files in a directory.

However, you may want to create a custom file store so that the file store's data is persisted to a particular storage device or when you want a JMS service that accesses a file store to be able to migrate with the store to another server member in a cluster. When configuring a file store directory, the directory must be accessible to the server instance on which the file store is located.

A JDBC store can be configured when a relational database is used for storage. A JDBC store enables you to store persistent messages in a standard JDBC-accessible database, which is accessed through a designated JDBC data source. The data is stored in the JDBC store's database table, which has a logical name of `WLStore`. It is up to the database administrator to configure the database for high availability and performance. JDBC stores also support migratable targets for automatic or manual JMS service migration.

See When to Use a Custom Persistent Store.

When to Use a Custom Persistent Store

WebLogic Server provides configuration options for creating a custom file store or JDBC-accessible store.

For example, you may want to:

- Place a file store's files on a particular device.
- Use a JDBC store rather than a file store for a particular server instance. If you want to persist transaction logs, use a JDBC TLOG store. See Using a JDBC TLog Store.
- Allow all physical stores in a cluster to share the same logical name.
- Logically separate different services to use different files or tables. (This may simplify administration and maintenance at the expense of reduced performance.)
- Migratable JMS-related services cannot use the default persistent store, so you must configure a custom store and target it to the same migratable target as the migratable JMS service. See Service Migration in *Administering Clusters for Oracle WebLogic Server*.

Methods of Creating a Custom Persistent Store

This section describes the different methods to configure a custom persistent store.

A user-defined persistent store can be configured in the following ways:

- Directly edit the configuration file (`config.xml`) of the server instance that is hosting a persistent store.
- Use the WebLogic Java Management Extensions (JMX) to create persistent stores. JMX is the Jakarta EE solution for monitoring and managing resources on a network. See *Developing Custom Management Utilities Using JMX for Oracle WebLogic Server*.
- Use the WebLogic Scripting Tool (WLST) to create persistent stores. WLST is a command-line scripting interface that you use to interact with and configure WebLogic Server instances and domains. See Understanding the WebLogic Scripting Tool.
- Use the WebLogic Configuration Wizard to change the options of the default persistent store. For detailed information on how to use the Configuration Wizard to configure a persistent store, see Creating a WebLogic Domain in *Creating WebLogic Domains Using the Configuration Wizard*.

Modifying Custom Persistent Store Parameters

Modifying certain custom store configuration options, such as a JDBC store's prefix or a file store's directory, do not necessarily require a server restart as described in the following procedures.

1. Set the targets of any dependent services to null (such as a JMS server that uses the custom store), and then setting the custom store target to null. (Setting a service's target to null implicitly shuts down the service.)
2. Reverse the process by setting the custom store target back to its original value and then setting the dependent resource targets back to their original values.

In cases where the custom store and JMS servers share a migratable target, you can administratively restart the migratable target.

4

Using Custom File Stores

This chapter explains how to configure the custom file stores for WebLogic Server. It includes the following sections:

Creating a Custom (User-Defined) File Store

The following section provides an example of a custom file store and configuration guidelines for choosing a synchronous write policy.

To create a custom file store, you can directly modify the default FileStoreMBean parameters. For instructions on using the WebLogic Remote Console to create a custom file store, see Create a File Store in the *Oracle WebLogic Remote Console Online Help*.

Main Steps for Configuring a Custom File Store

This section describes the main steps to create a custom file store.

Perform the following steps to create a custom file store:

1. Create a directory where the file store's data will be persisted.
2. Make a note of the following information:
   - For stores that may migrate, such as cluster targeted stores with a `Migration Policy != Off`, stores that are targeted to a migratable target, or stores hosted on a WebLogic Server JVM that may be moved to a different machine, always configure a directory that is centrally accessible from any location that the store may migrate to (do not leave the directory name at the default). This is necessary for migrated stores to recover any data written before the migration - including, for example, queued JMS messages. See Whole Server Migration and Service Migration in *Administering Clusters for Oracle WebLogic Server*.
   - For information about default, absolute, and relative file locations, see File Locations.
3. Associate the custom file store with the subsystem(s) or migratable target that will be accessing it, such as:
   - For JMS servers, select the custom file store on the General Configuration page.
   - For Store-and-Forward agents, select the custom file store on the General Configuration page.
   - For a Path Service, select the custom file store on the General Configuration page.

Example of a Custom File Store

This section provides an example of how a custom file store may look in a domain's configuration file with its files kept in a `/disk1/jmslog` directory.

```
<file-store>
  <name>SampleFileStore</name>
  <target>myserver</target>
```

```
  <directory>/disk1/jmslog</directory>
</file-store>
```

Table 4-1 briefly describes the file store configuration parameters that can be modified.

Table 4-1 Custom File Store Configuration Options

Option	Required	What It Does
Name	Yes	The name of the file store, which must be unique across all stores in the domain.
Targets	Yes	The server instance, cluster, or migratable target where a file store is targeted. Multiple subsystems can share the same file store, as long as they are all targeted to the same server instance or migratable target. **Note:** • When using a cluster to host a JMS Server, you must target the file store to the same cluster used by the JMS Server. See Simplified JMS Cluster and High Availability Configuration in *Administering JMS Resources for Oracle WebLogic Server*. • When using migratable targets for JMS services, you must target the file store to the same migratable target used by the JMS service. See Service Migration in *Administering Clusters for Oracle WebLogic Server*.
Directory	Yes	The path name to the directory on the file system where the file store is kept. **Note:** • For stores that may migrate, such as cluster targeted stores with a Migration Policy != Off, stores that are targeted to a migratable target, or stores hosted on a WebLogic Server JVM that may be moved to a different machine, always configure a directory that is centrally accessible from any location that the store may migrate to (do not leave the directory name at the default). This is necessary for migrated stores to recover any data written before the migration - including, for example, queued JMS messages. See Whole Server Migration and Service Migration in *Administering Clusters for Oracle WebLogic Server*. • For information about default, absolute, and relative file locations, see File Locations. • Modifying an existing file store's directory does not necessarily require a server restart, as described in Modifying Custom Persistent Store Parameters.
CacheDirectory	No	This setting only applies for the `Direct-Write-With-Cache` file store synchronous write policy. See Guidelines for Configuring a Synchronous Write Policy.
Logical Name	No	Optionally used with subsystems, like EJBs, when deploying a module to an entire cluster, but also require a different physical store on each server instance in the cluster. In such a configuration, each physical store would have its own name, but all the persistent stores would share the same logical name.
Synchronous Write Policy	No	Defines the IO behavior of a file store including immediate durability of individual write operations. Values are: Direct-Write (default), Direct-Write-With-Cache, Cache-Flush, and Disabled. See Guidelines for Configuring a Synchronous Write Policy.

For instructions on configuring a custom file store using the WebLogic Remote Console, see Create a File Store in the *Oracle WebLogic Remote Console Online Help*.

Guidelines for Configuring a Synchronous Write Policy

There are several Synchronous Write Policies available for file stores. The Synchronous Write Policy determines the behavior of the write operation of the file store.

You should select a policy that best suits your environment and meets your needs for runtime performance and data integrity after a possible crash. See Tuning the WebLogic Persistent Store in *Tuning Performance of Oracle WebLogic Server* for more details about tuning and performance specifics of Synchronous Write Policy and other file store options.

> **Note**
>
> To view a running custom or default file store's synchronous write policy and driver, check the `WL-280008 and WL-280009` messages in the server log.

Direct-Write-With-Cache Policy

For most scenarios, Oracle recommends using the `Direct-Write-With-Cache` policy. When this policy is selected, WebLogic Server writes synchronously to a primary set of files in the location defined by the `Directory` attribute of the file store configuration using a native I/O `wlfileio` driver. WebLogic Server also asynchronously writes to a corresponding cache file in the location defined by the `CacheDirectory` attribute of the file store configuration, which is done implicitly by using OS memory caching the cache file blocks as output buffers for the primary data file. The cache files are used for performance optimizations at runtime and boot time and for recovery. This combination of direct writing with a native file driver and the use of corresponding cache files typically provides the best overall performance with the most safe disk writes.

This option uses approximately twice as much disk space as other policies and stores files in two locations. You may need to consider disk space allocations in these locations and you may need to secure both of these locations.

When configuring file locations with the `Direct-Write-With-Cache` policy, the location of the `CacheDirectory` attribute should be a local directory, even when configuring for high availability (Whole Server Migration or Automatic Service Migration). The cache files are used for performance optimizations only. The true persistent storage for messages is defined by the `Directory` attribute of the file store configuration. Only that directory needs to be available to the migrated WebLogic Server instance or JMS service after migration. The same applies to disaster recovery scenarios: only the files defined in the `Directory` location need to replicated to the backup site.

> **Note**
>
> If the file store native `wlfileio` driver cannot be loaded, the store automatically runs in an alternate specialized `Direct-Write` policy mode. To view a running custom or default file store's configured and actual synchronous write policy and driver, examine the server log for `WL-280008` and `WL-280009` messages.
>
> Certain older versions of Microsoft Windows may incorrectly report storage device synchronous write completion if the Windows default `Write Cache Enabled` setting is used. This violates the transactional semantics of transactional products (not specific to Oracle), including file stores configured with a `Direct-Write` (default) or `Direct-Write-With-Cache` policy, as a system crash or power failure can lead to a loss or a duplication of records/messages. One of the visible symptoms is that this problem may manifest itself in high persistent message/transaction throughput exceeding the physical capabilities of your storage device. You can address the problem by applying a Microsoft supplied patch, disabling the Windows `Write Cache Enabled` setting, or by using a power-protected storage device.

Direct-Write Policy

When the `Direct-Write` policy is selected, WebLogic Server writes synchronously to a primary set of files in the location defined by the `Directory` attribute of the file store configuration using a native I/O wlfileio driver. This policy typically performs slower than the `Direct-Write-With-Cache` policy, but it uses less disk space and may have fewer environmental considerations to manage. The `Direct-Write` policy is typically faster than the `Cache-Flush` policy.

> **Note**
>
> Certain older versions of Microsoft Windows may incorrectly report storage device synchronous write completion if the Windows default `Write Cache Enabled` setting is used. This violates the transactional semantics of transactional products (not specific to Oracle), including file stores configured with a `Direct-Write` (default) or `Direct-Write-With-Cache` policy, as a system crash or power failure can lead to a loss or a duplication of records/messages. One of the visible symptoms is that this problem may manifest itself in high persistent message/transaction throughput exceeding the physical capabilities of your storage device. You can address the problem by applying a Microsoft supplied patch, disabling the Windows `Write Cache Enabled` setting, or by using a power-protected storage device.

Cache-Flush Policy

When the `Cache-Flush` policy is selected, WebLogic Server enables the default file write behavior of the operating system and storage device, which typically includes caching and scheduling file writes, but forces a flush of the cache to disk before completing a transaction. Transactions cannot complete until all of their writes have been flushed down to disk. This policy is reliable and scales well as the number of simultaneous users increases. It is transactionally safe, but tends to provide lower runtime performance than the direct-write policies in typical use cases, except in those cases with large numbers of simultaneous producers or consumers.

Disabled Policy

When the `Disabled` policy is selected, WebLogic Server relies on the default file write behavior of the operating system and storage device. In most cases, file writes are cached in memory and are scheduled for writing instead of being directly written to disk. The `Disabled` policy generally improves file store performance, often quite dramatically, but at the expense of possibly losing sent messages or generating duplicate received messages (even if messages are transactional) in the event of an operating system crash or a hardware failure. This is because transactions are complete as soon as their writes are cached in memory, instead of waiting for the writes to successfully reach the disk. Simply shutting down an operating system or killing a WebLogic Server process does not generate these failures, as an OS flushes all outstanding writes under these circumstances during a normal shutdown. Instead, these failures can be emulated by abruptly shutting the power off to a busy server.

5

Using a JDBC Store

This chapter explains how to configure and monitor the WebLogic Server persistent store, which provides a built-in, high-performance storage solution for WebLogic Server subsystems and services that require persistence.

Creating JDBC-accessible Stores

The following sections provide information on how to configure and use JDBC-accessible stores.

- JDBC TLog Stores: to persist transaction logs (TLOGs) in a database. See Using a JDBC TLog Store.
- JDBC Stores: to persist WebLogic Server instance services and subsystem information, excluding TLOGs, in a database. See Using a JDBC Store.

Using a JDBC TLog Store

You can configure a JDBC TLOG store to persist transaction logs to a database, which provides the benefits described in this section.

- Leverages replication and HA characteristics of the underlying database.
- Simplifies disaster recovery by allowing the easy synchronization of the state of the database and TLOGs.
- Improved Transaction Recovery service migration as the transaction logs to do not need to be migrated (copied) to a new location.

Main Steps for Configuring a JDBC TLOG Store

The main steps for creating a JDBC TLOG store are as follows:

1. Create a JDBC data source, GridLink data source, or multi data source to interface with the JDBC store. See Choosing a Data Source.
2. Create a JDBC TLOG store and associate it with the JDBC data source, GridLink data source, or multi data source created in Step 1. .
3. Optional. It is highly recommended that you configure the Prefix option to a unique value for each configured JDBC TLOG store.
4. For high availability, make your data source available to backup servers. See Server Migration when using a JDBC TLOG Store.

Choosing a Data Source

You can choose one of the following data source types, depending on your WebLogic Server license and application needs:

- Generic Data Sources—See Creating a JDBC Data Source in *Administering JDBC Data Sources for Oracle WebLogic Server*.

- GridLink Data Sources—See Using GridLink Data Sources in *Administering JDBC Data Sources for Oracle WebLogic Server*.
- Multi data sources—Backed by a fully replicated, zero-latency database, such as Oracle RAC. See Configuring JDBC Multi Data Sources and Using Multi Data Sources with Oracle RAC in *Administering JDBC Data Sources for Oracle WebLogic Server*.

Example of a JDBC TLOG Store

Here's an example of how a JDBC TLOG store may look in the configuration file, using the JDBC data source `MyDataSource`, and with a logical name specified:

```
<server>
  <transaction-log-jdbc-store>
    <data-source>MyDataSource</data-source>
    <prefix-name>TLOG_MS1</prefix-name>
    <create-table-ddl-file>myDDL/myCreateTable.sql</create-table-ddl-file>
    <max-retry-seconds-before-tlog-fail>120</max-retry-seconds-before-tlog-fail>
  </transaction-log-jdbc-store>
</server>
```

Table 5-1 describes the JDBC TLOG store configuration parameters that can be modified.

Table 5-1 JDBC TLOG Store Configuration Options

Option	Required	What it Does
Prefix Name	No	The prefix for the JDBC store's table is generally entered in the following format: `[[[catalog.]schema.]prefix]` When using multiple JDBC stores, it is required to set this option to a unique value for each configured JDBC store. When no prefix is specified, the JDBC store table name is simply `WLStore` and the database implicitly determines the schema according the current user of the JDBC connection. Also, two JDBC stores cannot share the same database table. See Using Prefixes with a JDBC Store. Modifying an existing JDBC store's prefix does not necessarily require a server restart, as described in Modifying Custom Persistent Store Parameters.
Create Table from DDL File	No	Optionally used with supported DDL (data definition language) files to create the JDBC store's database table (`WLStore`). This option is ignored when the JDBC store's database table already exists. See Creating a JDBC Store Table Using Default and Custom DDL Files.
Deletes Per Batch Maximum	Default is 20	The maximum number of table rows that are deleted per database call.
Inserts Per Batch Maximum	Default is 20	The maximum number of table rows that are inserted per database call.
Deletes Per Statement Maximum	Default is 20	The maximum number of table rows that are deleted per database call.

Table 5-1 (Cont.) JDBC TLOG Store Configuration Options

Option	Required	What it Does
MaxRetrySecondsBeforeTLogFail	Default is 300	The maximum amount of time, in seconds, WebLogic Server tries to recover from a JDBC TLog store failure. If store remains unusable after this period, WebLogic Server set the health state to HEALTH_FAILED. A value of 0 indicates WebLogic Server does not conduct a retry and and immediately sets the health state as HEALTH_FAILED.
MaxRetrySecondsBeforeTXRollback	Default is 60	The maximum amount of time, in seconds, WebLogic Server waits before trying to recover from a JDBC TLog store failure while processing a transaction. If store remains unusable after this amount of time, WebLogic Server rolls back the affected transaction. A value of 0 indicates WebLogic Server does not conduct a retry and rolls back the transaction immediately. The practical maximum value is a value less than the current value of `MaxRetrySecondsBeforeTLogFail`.
RetryIntervalSeconds	Default is 5	The amount of time, in seconds, WebLogic Server waits before attempting to verify the health of the TLOG store after a store failure has occurred.
N/A	1000	The amount of time, in milliseconds, a JDBC Store reconnect retry or a TLOG-in-DB Store attempts to re-establish a connection to a database, before the Store shuts down, and all the operations waiting on the Store are unblocked.The minimum value that can be configured through the `ReconnectRetryPeriodMillis` is 200, and the maximum value is 300000. For more information about JDBC Store reconnect retry, see Configuring JDBC Store Reconnect Retry.
ReconnectRetryIntervalMillis	200	The amount of time in milliseconds, between reconnect attempts, during the connection retry period. The minimum value that can be configured through the `ReconnectRetryIntervalMillis` is 100, and the maximum value is 10000. See Configuring JDBC Store Reconnect Retry.

For instructions on configuring a JDBC TLOG store using the WebLogic Server Administration Console, see Configure the Transaction Log Store in the *Oracle WebLogic Server Administration Console Online Help*.

Configuration Guidelines

The following section provides guidelines for configuring JDBC TLOG stores.

- Only globally-scoped (not application-scoped) data sources can be targeted to a JDBC TLOG store.
- Only one JDBC TLOG store can be configured per WebLogic Server. Conversely, multiple WebLogic Servers can not share a JDBC TLOG store.
- You must configure a JDBC TLOG store. The default is to persist TLOG information to the server's default persistent store.

- You cannot use a data source that is configured to use an XA JDBC driver or is configured to support global transactions. Use a non-XA data source.
- For general rules on JDBC-accessible stores, see Guidelines for Configuring a JDBC Store.

Additional Considerations

The following section provides additional information on JDBC TLOG store behavior and limitations:

- The database used to store the TLOG information must be available at server startup. If the database is not available, the WebLogic Server instance will fail to boot.
- Only the JTA sub-system can use the JDBC TLOG store to persist information about committed transactions coordinated by the server that may not have been completed. No other systems can access the JDBC TLOG store.
- Using a JDBC TLOG store does not change LLR behavior. A JDBC TLOG store can be used with or without LLR. When used in tandem with LLR transactions, the transaction committing information is stored in a LLR table but the checkpoint records and heuristic logs are stored in the JDBC TLOG store.
- If the TLOG store is changed from one store type to another or from one location to another, the change takes effect only after reboot and all pending transactions in the old store are not be copied to the new store. You must ensure there are no pending transactions before changing the TLOG store type or location.
- If the JDBC TLOG store becomes unavailable, the JTA health state transitions to `FAILED` and any global transactions will fail. In turn, the server life-cycle changes to `FAILED`. The JTA Transaction Recovery System then attempts to recover from transient runtime errors if possible and resolves any in-doubt transactions. See Server Migration when using a JDBC TLOG Store.
- If the database used to store TLOG is corrupted and can not be restored, than all pending transaction information is lost.
- If database tables or rows used by the JDBC TLOG store are locked for some reason in the database, the database administrator must resolve these locks manually. Otherwise, the JTA subsystem is blocked and will be suspended until the lock(s) are released, or encounters an exception due to lock. The JTA subsystem will remain unable to operate correctly until the lock(s) are released or the value of MaxRetrySecondsBeforeTLOGFail is exceeded.

> **Note**
>
> Different databases have different features for locked local transactions. Some databases may have trouble resolving database locks in a timely manner. You may need to contact your database administrator for more information on basic row locking issues that may occur in your application environment.

Server Migration when using a JDBC TLOG Store

WebLogic Server supports both manual and automatic migration of the Transaction Recovery Service when using a JDBC TLOG store. The data source used by the JDBC TLOG store must be targeted on both the primary server instance and a backup server instance. Oracle

recommends targeting the data source to all the server instances of a cluster. See Transaction Recovery After a Server Fails in *Developing JTA Applications for Oracle WebLogic Server*.

Monitoring a JDBC TLOG Store

You can monitor statistics for Transaction Log Store statistics and for each open store connection. For general information on how to monitor persistent stores, see Monitoring the WebLogic Persistent Store.

How to Monitor the JDBC TLOG Store Health State

When you configure WebLogic Server to use a JDBC TLOG store, the store is registered with the Health system as a non-critical subsystem using a name with the following pattern:

`PersistentStore.TLOG_servername`

where *servername* is the name of the WebLogic Server instance hosting the primary TLOG store.

How to Monitor Transaction Log Store Statistics

You can monitor Transaction Log Store statistics in the WebLogic Remote Console, see View Transaction Statistics in *Oracle WebLogic Remote Console Online Help*.

How to Monitor Transaction Log Store Connections

You can monitor Transaction Log Store connection statistics in the WebLogic Remote Console, see View Transaction Statistics in *Oracle WebLogic Remote Console Online Help*.

Security Considerations

Properly secure your application environment, including the JDBC TLOG store table. Failure to do so may allow a process to:

- Delete information, maliciously or unintentionally. Such a deletion can cause transaction information to be lost and cause affected global transactions to be completed heuristically.
- Modify information, maliciously or unintentionally. Such modification can cause unexpected behavior.
- Read confidential transaction information, such as when transaction starts and what resources are involved.
- Access the database instance or database machine.
- Access the network between JTA and the database, potentially intercepting, viewing, or modifying data.

See Secure File Store Data.

Using a JDBC Store

The following sections provide an example of a JDBC store, and information about creating a database table for a JDBC store, either using an existing DDL, a custom DDL, and using Oracle blob record columns in a DDL file.

To create a JDBC store, you can directly modify the default JDBCStoreMBean parameters. For instructions on using the WebLogic Remote Console to create a JDBC store, see Create a JDBC Store in *Oracle WebLogic Remote Console Online Help*.

For configuration guidelines on using prefixes with JDBC stores and recommended JDBC data source settings, see Guidelines for Configuring a JDBC Store.

Main Steps for Configuring a JDBC Store

The main steps for creating a JDBC store are as follows:

1. Create a JDBC data source or multi data source to interface with the JDBC store.
2. Create a JDBC store and associate it with the JDBC data source or multi data source.
3. It is highly recommended that you configure the Prefix option to a unique value for each configured JDBC store table.
4. Associate the JDBC store with the subsystem(s) that will be using it, such as:
   - For JMS servers, select the JDBC store on the General Configuration page.
   - For Store-and-Forward agents, select the JDBC store on the General Configuration page.
   - For a Path Service, select the custom file store on the General Configuration page.

Example of a JDBC Store

Here's an example of how a JDBC store may look in the configuration file, using the JDBC data source `MyDataSource`, and with a logical name specified:

```
<jdbc-store>
  <name>SampleJDBCStore</name>
  <target>yourserver</target>
  <data-source>MyDataSource</data-source>
  <logical-name>Baz</logical-name>
</jdbc-store>
```

Table 5-2 describes the JDBC store configuration parameters that can be modified.

Table 5-2 JDBC Store Configuration Options

Option	Required	What It Does
Name	Yes	The name of the JDBC store, which must be unique across all stores in the domain.

Table 5-2 (Cont.) JDBC Store Configuration Options

Option	Required	What It Does
Targets	Yes	The server instance, cluster, or migratable target where a JDBC store is targeted. Multiple subsystems can share the same JDBC store, as long as they are all targeted to the same server instance or migratable target. **Note:** • When using a cluster to host a JMS Server, you must target the JDBC store to the same cluster used by the JMS Server. See Configuring Dynamic Clustered JMS in *Administering JMS Resources for Oracle WebLogic Server*. • When using migratable targets for JMS services, you must target the JDBC store to the same migratable target used by the JMS service. See Service Migration in *Administering Clusters for Oracle WebLogic Server*.
Data Source	Yes	The JDBC data source or multi data source used by this JDBC store to access the store's database table (`WLStore`). This data source or multi data source must be targeted to the same target as the JDBC store. **Note:** The JDBC store must use a JDBC data source that uses a non-XA JDBC driver and has **Supports Global Transactions** disabled. This limitation does not remove the XA capabilities of layered subsystems that use JDBC stores. For example, WebLogic JMS is fully XA-capable regardless of whether it uses a file store or any JDBC store.
Prefix Name	No	The prefix for the JDBC store's table is generally entered in the following format: `[[[catalog.]schema.]prefix]` When using multiple JDBC stores, it is required to set this option to a unique value for each configured JDBC store. When no prefix is specified, the JDBC store table name is simply `WLStore` and the database implicitly determines the schema according the current user of the JDBC connection. Also, two JDBC stores cannot share the same database table. See Using Prefixes with a JDBC Store. Modifying an existing JDBC store's prefix does not necessarily require a server restart, as described in Modifying Custom Persistent Store Parameters.
Logical Name	No	Optionally used with WebLogic Server subsystems, like EJBs, when deploying a module to an entire cluster, but also require a different physical store on each server instance in the cluster. In such a configuration, each physical store would have its own name, but all the persistent stores would share the same logical name.
Create Table from DDL File	No	Optionally used with supported DDL (data definition language) files to create the JDBC store's database table (`WLStore`). This option is ignored when the JDBC store's database table already exists. See Creating a JDBC Store Table Using Default and Custom DDL Files.

For instructions on configuring a JDBC store using the WebLogic Remote Console, see Create a JDBC Store in *Oracle WebLogic Remote Console Online Help*.

Supported JDBC Drivers

When using a JDBC store, the backing database can be any database that is accessible through a JDBC driver. WebLogic Server detects some drivers for supported databases.

For each of these databases, there are corresponding DDL (data definition language) files within the *ORACLE_HOME*\wlserver\modules\com.bea.core.store.jdbc_1.0.0.0.jar file, in the weblogic/store/io/jdbc/ddl directory, where *ORACLE_HOME* is the top-level installation directory of your WebLogic Server installation.

Table 5-3 Supported JDBC Drivers and Corresponding DDL Files

Database	DDL Files
IBM DB2	db2.ddl db2v6.ddl
Informix	informix.ddl
Microsoft SQL (MSSQL) Server	mssql.ddl
MySQL	mysql.ddl
Oracle	oracle.ddl oracle_blob.ddl oracle_blob_securefile.ddl
Sybase	sysbase.ddl

The DDL files are actually text files containing the SQL commands (terminated by semicolons) that create the JDBC store's database table (WLStore). Therefore, if you are using a database that is not included in this list, you can copy and edit any one of the existing DDL files to suit your specific database, as described in Creating a JDBC Store Table Using a Custom DDL File.

Creating a JDBC Store Table Using Default and Custom DDL Files

The JDBC Store Configuration page provides an optional Create Table from DDL File option, through which you can access a pre-configured DDL file that is used to create the JDBC store's backing table (WLStore). This option is ignored when the JDBC store's backing table already exists. It is mainly used to specify a custom DDL file created for an unsupported database, or when upgrading JMS JDBC store tables from a prior release to a current JDBC Store table.

If a DDL file name is *not* specified in the Create Table from DDL File field, and the JDBC store detects that its backing table does not already exist, the JDBC store automatically creates the table by executing a pre-configured DDL file that is specific to the database vendor (see Supported JDBC Drivers).

If a DDL file name is specified in the Create Table from DDL File field, and the JDBC store detects that its backing table does not already exist, the JDBC store searches for the specified DDL file in the file path first, and then, if not found, searches for the DDL file in the CLASSPATH. Once found, the SQL within the DDL file is executed to create the JDBC store's backing table. If the configured file is not found and the table doesn't already exist, the JDBC store will fail to boot.

Creating a JDBC Store Table Using a Custom DDL File

To use a different database from those listed in Supported JDBC Drivers, you can copy and edit any one of the existing DDL template files to suit your specific database.

1. Use the JAR utility supplied with the JDK to extract the DDL files to the `/weblogic/store/io/jdbc/ddl` directory using the following command:

   ```
   jar xf com.bea.core.store.jdbc_1.0.0.0.jar /weblogic/store/io/jdbc/ddl
   ```

   > **Note**
   >
   > If you omit the `weblogic/store/io/jdbc/ddl` parameter, the entire jar file is extracted.

2. Edit the DDL file for your database. An SQL command can span several lines and is terminated with a semicolon (;). Lines beginning with pound signs (#) are comments.
3. Save your changes and rename the new DDL appropriately (for example, `mydatabase.ddl`)
4. Create a JDBC store, as explained in Create a JDBC Store in *Oracle WebLogic Remote Console Online Help*.
5. Use the Create Table from DDL File option on the General Configuration page to specify your custom DDL file (for example, `/mydatabase.ddl`).

   > **Note**
   >
   > On Windows systems, for full path names always include the drive letter.

Enabling Oracle BLOB Record Columns

For Oracle databases, you can use the `oracle_blob.ddl` or `oracle_blob_securefile.ddl` file to create a JDBC store table with a BLOB record column type rather than the default `LONG RAW` record column type. The `oracle_blob.ddl` is used to create Oracle basic file BLOBs and the `oracle_blob_securefile.ddl` file is used to create Oracle secure file BLOBs. Both files types are pre-configured and supplied in the WebLogic `CLASSPATH`, as described in Supported JDBC Drivers.

Oracle Database 11g Release 2 includes a zero copy I/O performance enhancement for Secure Files and a logical cache for BLOBs. Use of these enhancements can improve throughput with a JDBC store when message sizes are large and when network connections to the database are slow. The Oracle `LONG RAW` datatype is typically better performing than BLOBS when using a fast connection to the database.

> **Note**
>
> If you need to preserve data already in a Oracle `LONG RAW` column, but still want to switch the column to BLOB, *do not* use this method. Instead, consult the Oracle documentation for the `SQL ALTER TABLE` command.

To use the Oracle BLOB DDL with a JDBC store:

1. Shut down the server instance that uses the JDBC store.
2. Delete the current JDBC table, as explained in Managing JDBC Store Tables.
3. Reboot the server instance.

4. Create a new JDBC store.
5. In the `Create Table from DDL File` field on the General Configuration page, enter the location of:
   - the `oracle_blob.ddl` file as: `/weblogic/store/io/jdbc/ddl/oracle_blob.ddl`
   - the `oracle_blob_securefile.ddl` file as: `/weblogic/store/io/jdbc/ddl/oracle_blob_securefile.ddl`
6. Click **Finish** to create the JDBC store's backing table.

When using Oracle BLOBS, you may be able to improve performance by tuning the `ThreeStepThreshold` value.

When the JDBC store schema contains an Oracle BLOB column (basic file or secure file), the JDBC store populates the BLOB data using one of the following implementations based on the size of the BLOB data:

- The JDBC store inserts a row with BLOB data directly into the store table in a single step. Because only a single step is involved, JDBC batch insert is also adopted and performs best when the data size is small. This implementation is used when the BLOB data to be inserted is less than or equal to the value of the `ThreeStepThreshold`.
- The JDBC store inserts a row with BLOB data into the store table in three steps using the Oracle LOB API. This implementation provides better performance when the data size is large. This implementation is used when the BLOB data to be inserted is greater than the value of the `ThreeStepThreshold`.

The default value of `ThreeStepThreshold` is 200K.

Managing JDBC Store Tables

The JDBC `utils.Schema` utility allows you to regenerate a new JDBC store database table (`WLStore`) by deleting the existing version. Running this utility is usually not necessary, since WebLogic Server automatically creates this table for you. However, if your existing JDBC store database table somehow becomes corrupted, you can delete it using the `utils.Schema` utility.

The `utils.Schema` utility is a Java program that takes command-line arguments to specify the following:

- JDBC driver
- Database connection information
- Name of a file containing the SQL Data Definition Language (DDL) commands that create the database table

Using the utils.Schema Utility to Delete a JDBC Store Table

Enter the `utils.Schema` command, as follows:

```
$ java utils.Schema url JDBC_driver [options] DDL_file
```

> **Note**
>
> To execute `utils.Schema`, your `CLASSPATH` must contain the `weblogic.jar` file.

Table 5-4 lists the `utils.Schema` command-line arguments.

Table 5-4 Command-line arguments for utils.Schema

Argument	Description
`url`	Database connection URL. This value must be a colon-separated URL as defined by the JDBC specification.
`JDBC_driver`	Full package name of the JDBC Driver class.

`options`	Optional command options.
	If required by the database, you can specify:
	• The user name and password as follows:
	`-u <username> -p <password>`
	• The Domain Name Server (DNS) name of the JDBC database server as follows:
	`-s <dbserver>`
	You can also specify the `-verbose` option, which causes `utils.Schema` to echo SQL commands as they are executed.

`DDL_file`	The full pathname of the DDL text file containing the SQL commands that you want to execute. For more information, see Supported JDBC Drivers.

For example, the following command deletes a JDBC table named `MYWLStore` in an Oracle server named `DEMO`, with the user name `user1` and password `foobar`:

```
$ echo "drop MYWLStore;" > drop.ddl

$ java utils.Schema
jdbc:weblogic:oracle:DEMO \
weblogic.jdbc.oci.Driver -u user1 -p foobar -verbose \
drop.ddl
```

Configuring JDBC Store Reconnect Retry

The JDBC Store reconnect retry period indicates the time period when a WebLogic JDBC Store or a TLOG in-DB Store retries to connect to a database, before the Store shuts down and requires a restart. You can configure the Store retry period through Command line system properties, MBeans and WLST.

Using WLST and JMX MBeans

The JDBC Store reconnect retry period controls the length of the time period required by a JDBC Store reconnect retry or a TLOG-in-DB Store to retry database requests before a Store shuts down, and requires a restart. The JDBC Store reconnect retry interval controls the length of the time in milliseconds between reconnect attempts during the connection retry period. You can configure the JDBC Store reconnect retry period and interval by using the `ReconnectRetryPeriodMillis` and `ReconnectRetryIntervalMillis` attributes available in the `JDBCStoreMBean` and `TransactionLogJDBCStoreMBean`. For more information about the Retry attributes, see MBean Reference for Oracle WebLogic Server.

Using Command Line System Properties

You can configure the retry period and interval for custom JDBC Store reconnect retry and TLOG-in-DB Stores by setting system properties on the WebLogic Server command line. The `-DwebLogic.store.jdbc.RecnnectRetryPeriodMillis=<millis>` and `-`

`Dweblogic.store.jdbc.ReconnectRetryIntervalMillis=<millis>` option specifies the JDBC Store reconnect retry in period and interval available in the WebLogic Server domain.

The `-Dweblogic.store.jdbc.ReconnectRetryPeriodMillis=<millis>` system property overrides legacy properties`-Dweblogic.store.jdbc.IORetryDelayMillis=<millis>` or `-Dweblogic.jms.store.JMSJDBCIORetryDelay=<seconds>`, if they are set on the same command line. If the retry period property is not set, then the legacy properties will take effect.

Deprecation Note: All JDBC Store reconnect retry command line properties are deprecated as of 12.2.1.0 and may be removed in a future release. In 12.2.1.0 and later, Oracle recommends setting these values through MBean attributes instead.

Important Tuning Considerations for Reconnect Retry

It is important to consider the following when configuring a JDBC Store reconnect retry period:

- The total elapsed time before a Store fails may sometimes be more than double the configured retry period.
- It is advisable to configure more tolerant retry periods of up to 15 seconds or more instead of the maximum, since a retry period that is set to too long may lock up WebLogic Server applications and client applications during this period.
- JDBC Store reconnect retry tuning should be configured to align with transaction tuning.
  - Consider tuning JTA transaction time outs to be higher than the retry period so that the application transactions that involve a JDBC Store do not time out waiting for a JDBC Store to successfully recover from a database failure. The default transaction time out for a domain is 30 seconds and is tunable via the `TimeoutSeconds` attribute on the `JTAMBean`. In addition, transaction time outs are tunable on a per application basis.
  - WebLogic's transaction system will temporarily stop allowing a JDBC Store to participate in transactions if the JDBC Store is unresponsive for more than the `JTAMBean MaxXACallMillis` attribute (default is 1200000 millis/two minutes). Once JTA decides a Store is unresponsive, it will not attempt to allow the Store to participate in transactions until after `MaxResourceUnavailableMillis` has passed (default is 1800000 millis/30 minutes). It is therefore advisable to ensure that `MaxXACallMillis` is at least twice the JDBC Store reconnect retry period.
  - If the retry period is configured on a TLOG-in-DB Store, it should be set to less than half of the `TransactionLogJDBCStoreMBean MaxRetrySecondsBeforeTLOGFail` setting. otherwise, the TLOG-in Store may delay failure longer than the `TLOGFail` setting.
- A JDBC Store reconnect retry period is configured in milliseconds while some transaction settings are configured in seconds.

Configuring a JDBC Store Connection Caching Policy

By default, a WebLogic JDBC Store obtains two JDBC connections from its Data Source, and caches these connections throughout a Store's lifetime. You can optionally tune the JDBC Store's Connection Caching Policy to reduce the number of cached JDBC connections.

Using WLST and JMX MBeans

The JDBC Store Connection Caching Policy setting controls how many JDBC connections it caches. The Connection Caching policy can be configured by using the `ConnectionCachingPolicy` attribute available in the `JDBCStoreMBean` and `TransactionLogJDBCStoreMBean`. The valid values for `ConnectionCachingPolicy` attribute are

- `DEFAULT`, `MINIMAL` and `NONE`. For more information about the valid values, see MBean Reference for Oracle WebLogic Server and Table 6-5.

> **Note**
>
> The `NONE` policy requires additional tuning to avoid deadlocking a server. For more information about tuning the `NONE` JDBC Store Connection Caching Policy, see Important Tuning Considerations for the NONE Connection Caching Policy.

Using a Command Line Parameter

You can configure the JDBC Store Connection Caching Policy by setting a system property on the WebLogic Server command line. The `-Dweblogic.store.jdbc.ConnectionCachingPolicy` option specifies the WebLogic JDBC Store Connections available in the WebLogic Server domain. For more information about the valid values that can be set for this Policy, see Table 6-5.

> **Note**
>
> - The `weblogic.store.jdbc.ConnectionCachingPolicy` system property has been deprecated as of 12.2.1.1.0, and may not remain available in future releases. Oracle recommends configuring a Connection Policy through WLST or MBeans instead. For more information about tuning the `NONE` JDBC Store Connection Caching Policy, see Important Tuning Considerations for the NONE Connection Caching Policy
> - The `NONE` policy requires additional tuning to avoid deadlocking a server. For more information about tuning the `NONE` JDBC Store Connection Caching Policy, see Important Tuning Considerations for the NONE Connection Caching Policy.

JDBC Store Connection Caching Behavior

A JDBC Store's Connection Caching behavior is determined by the combination of its Connection Caching Policy setting and Worker Count setting.

Table 5-5 JDBC Store Connection Caching Policy behavior

Connection Caching Policy	Cached Connections when Worker count=1	Cached Connections when WorkerCount>1	Description
DEFAULT	2	2+ Worker Count	By default, each JDBC Store instance caches two database connections for the life of store. If the JDBC Store worker-count is configured to be more than two, the store opens an additional connection for each worker.
MINIMAL	1	1+WorkerCount	Each JDBC store instance caches a single database connection for the life of the store. If the JDBC worker- count is configured to be two or higher, the store opens one connection for each worker. The performance of this setting may be less than `DEFAULT` for low concurrency messaging scenarios.

Table 5-5 (Cont.) JDBC Store Connection Caching Policy behavior

Connection Caching Policy	Cached Connections when Worker count=1	Cached Connections when WorkerCount>1	Description
NONE	0	N/A	Each JDBC store instance obtains a connection from its data source as needed, and releases the connection when finished. The `NONE` setting is not compatible with a JDBC Store worker-count of two or more, and will result in a configuration validation exception. The performance of this setting will be lesser than `DEFAULT` or `MINIMAL`. **Warning**: To avoid the risk of dead-locking a WebLogic Server, Oracle strongly recommends configuring a dedicated data source for NONE connection-caching-policy JDBC stores.

Important Tuning Considerations for the NONE Connection Caching Policy

It is important to consider the following tuning considerations when a JDBC Store Connection Caching Policy is set to `NONE`:

- Use a dedicated Data Source to avoid deadlocks - It is strongly advised to configure JDBC Stores to use a dedicated Data Source when the JDBC Store Connection Caching Policy is set to `NONE`. A`NONE` policy JDBC Store may deadlock a server or eventually fail if it is configured to share the Data Source with applications or non-store services.

  For example, consider an application that performs the following steps:

  1. Obtains a Data Source connection.
  2. Sends a persistent JMS message through a JMS Server with a NONE policy, JDBC Store that uses the same Data Source.
  3. Closes the Data Source connection.

- It is possible that step 1 can obtain the last available connection in the Data Source connection pool, and therefore in step 2, the JMS send call will block until the `NONE` policy JDBC Store is able to get a connection from the same pool. This is a problem because step 2 will potentially never get a connection no matter how long it waits since it is possible that all other applications are also blocked in step 2 (and therefore no application can get to step 3 in order to free up a connection). This problem in turn can cause a server or client to have many stuck threads and/or cause a JDBC Store to ultimately shutdown once it waits too long to try and get a connection.
- Tune a large enough Data Source connection pool - A JDBC Store uses multiple concurrent connections when its dependent services (such as JMS) first initialize. Hence, the Data Source pool must be configured so that it can grow somewhat larger than the number of JDBC Stores that are using the pool.
- Enable and tune Data Source connection testing - Enabling Data Source connection testing helps provide JDBC Store resiliency during database access failures. But, note that if performance is a concern, then frequent Data Source connection testing should be avoided, since it lowers performance of a `NONE` policy JDBC Store. In general, it is advisable to enable the Data Source `Test Connection` on `Reserve` setting, and set the Data Source `Seconds to Trust` and `Idle Pool Connection` value higher than zero, and lower than the JDBC Store `Connection Retry Interval Millis` value. See Connection

Testing Options for a Data Source in *Administering JDBC Data Sources for Oracle WebLogic Server*.

- Monitor and tune Prepared Statement Caching performance - A JDBC Store configured with `NONE` may yield poor performance if its Data Source or JDBC driver Prepared Statement Cache size is too small. To check if cache misses are lowering performance, monitor your prepared statement cache activity when under load. This monitoring should show frequent cache hits and few cache misses, but if you see many cache misses then increase your cache size.
- Monitor Oracle RAC with GridLink performance and potentially tune accordingly. If a `NONE` policy JDBC Store yields poor performance in comparison to other policies when using Oracle RAC with a GridLink driver, the potential workarounds are:
  - Use a Multi Data Source instead of GridLink Data Source.
  - Rebuild JDBC Store tables with a reverse index. See Rebuilding the Store Table Index for an Oracle Database.

> **Note**
>
> The `NONE` policy may yield measurably lower performance than the `MINIMAL` or `DEFAULT` policies even if all of the above considerations are carefully followed.

Guidelines for Configuring a JDBC Store

The following sections provide guidelines for using JDBC store prefixes, recommended WebLogic JDBC data source settings for JDBC stores, and handling JMS transactions with JDBC stores.

Using Prefixes with a JDBC Store

The JDBC store database contains a database table, named `WLStore`, that is generated automatically and is used internally by WebLogic Server. The JDBC store provides an optional Prefix Name parameter, which can be used to provide more precise access to the database table.

It is always a best practice to configure a prefix for the JDBC `WLStore` table name, especially when:

- The database requires fully-qualified names. (You should verify this with your database administrator.)
- There is more than one JDBC store instance sharing a database, since no two JDBC stores can share the same table.
- There are many tables in the database. Setting the prefix reduces the number of tables the JDBC store must search through to find its table during boot.

JDBC Store Table Rules

To avoid potential data loss, follow these rules:

- Each JDBC store table name must be unique.
- If multiple JDBC stores share a table, the behavior is undefined and data loss is likely.
- There is no procedure for combining two database tables into a single table.

Prefix Name Format Guidelines

For most databases, the Prefix Name option for the JDBC store's backing database table should be set in the following format for each configured JDBC store, which will result in a valid table name when prepended to the JDBC store table name:

```
[[[catalog.]schema.]prefix]
```

Note that each period in the `[[[catalog.]schema.]prefix]` format is significant. Generally, *`catalog`* identifies the set of system tables being referenced by the DBMS, and *`schema`* generally corresponds to ID of the table owner (*`username`*). When no prefix is specified, the JDBC store table name is simply `WLStore` and the database implicitly determines the schema according the current user of the JDBC connection.

For example, in a production database, the database administrator could maintain a unique table for the Sales department, as follows:

```
[[[Production.]JMSAdmin.]Sales]
```

The resulting table will be created in the Production catalog, under the JMSAdmin schema, and will be named `SalesWLStore`.

For some DBMS vendors, such as Oracle, there is no catalog to set or choose, so the format simplifies to `[[schema.]prefix]`. For more information, refer to your DBMS documentation for instructions on fully-qualified table names, but note that the syntax specified by the DBMS may differ from the format required for this option.

> ⚠ **Caution**
>
> If the Prefix Name setting is changed, but the WLStore database table already exists in the database, take care to preserve existing table data. In this case, the existing database table must be renamed by a database administrator to match the new configured table name.

Recommended JDBC Data Source Settings for JDBC Stores

The following settings are recommended when you use a JDBC data source or multi data source for JDBC stores.

Automatic Reconnection to Failed Databases

WebLogic Server provides robust JDBC data sources that can automatically reconnect to failed databases after they come back online, without requiring you to restart WebLogic Server. To take advantage of this capability, and make your use of JDBC stores more robust, configure the following options on the JDBC data source associated with the JDBC store:

```
TestConnectionsOnReserve="true"
TestTableName="SYSTABLES"
ConnectionCreationRetryFrequencySeconds="600"
```

For more information about JDBC default Test Table Names, see Connection Testing Options for a Data Source in the *Administering JDBC Data Sources for Oracle WebLogic Server*. For more information about setting the number of database reconnection attempts, see the Enabling Connection Creation Retries section in *Administering JDBC Data Sources for Oracle WebLogic Server*.

Required Setting for Oracle DB2 Type 4 JDBC Drivers

For data sources used as a JDBC store that use the Oracle Type 4 JDBC driver for DB2, the `BatchPerformanceWorkaround` property must be set to "true" due to internal JMS batching requirements.

Handling JMS Transactions with JDBC Stores

The JDBC store must use a JDBC data source that uses a non-XA JDBC driver and has **Supports Global Transactions** disabled. This limitation does not remove the XA capabilities of layered subsystems that use JDBC stores. For example, WebLogic JMS is fully XA-capable regardless of whether it uses a file store or any JDBC store.

Because the JDBC store implements the XAResource interface, it acts as it's own resource manager and handles the transactions above the JDBC driver level. That is, the store itself implements the `XAResource` and handles the transactions without depending on the database (even when the messages are stored in the database).

This means that whenever you are using a JDBC store and a database (even if it is the same database where the JMS messages are stored), then it is two-phase commit transaction.

For more information about using JMS transactions with a JDBC store, see Using Transactions with WebLogic JMS in *Developing JMS Applications for Oracle WebLogic Server*.

From a performance perspective, you may also boost your performance as follows:

- Ensure that the JDBC data source used for the database work exists on the same server instance as the JMS destination—the transaction will still be two-phase, but it will be handled with less network overhead.
- Use file stores rather than JDBC stores.
- Configure multiple services to share the same store if they will commonly be invoked within the same transaction.
- If an application directly performs database operations in addition to invoking store services (such as JMS) within the same transaction, consider using a JDBC data source with Logging Last Resource (LLR) enabled for the database operations.

  With the LLR optimization, the transaction will follow the two-phase commit protocol, but the database operations will be handled in a single local transaction, which may improve overall transaction performance. For more information on using the LLR optimization, see Understanding the Logging Last Resource Transaction Option in *Administering JDBC Data Sources for Oracle WebLogic Server*.

Enabling I/O Multithreading for JDBC Stores

Under heavy JDBC store I/O loads, you can improve performance by configuring a JDBC store to use multiple JDBC connections to concurrently process I/O operations.

> **Note**
>
> Enabling I/O multithreading under light loads may actually reduce performance. Oracle recommends that you tune your applications appropriately.

To enable I/O multithreading, set the `Worker Count` attribute to an integer value greater than 1. The default value of this configuration property is 1 and disables this option. The `Worker Count` attribute specifies the number of worker threads the JDBC store uses to process store I/O. Each worker thread acquires one JDBC connection from the configured data source pool when the store is opened. In many cases, benefits of multithreading tends to decrease after 4 concurrent threads. When using a slow connection to the database, multithreading may not improve performance.

> **Note**
>
> If you set the `Worker Count` to a value where there are not enough connections available in the connection pool, the JDBC store will fail to open.

You can tune the workload for each worker thread by changing the value of the `Worker Preferred Batch Size` attribute. Increasing the value of this attribute incrementally increases the workload assigned to each worker thread. The workload consists of store I/O requests, which are grouped and pushed to each JDBC worker thread for processing. If the size of individual I/O requests is commonly very large (for example, requests to store 1 MB JMS messages), then tune the value of `Worker Preferred Batch Size` to a smaller value for better performance.

Rebuilding the Store Table Index for an Oracle Database

When I/O multithreading is enabled, multiple JDBC connections are used to concurrently process store I/O operations which can result in database resource contention. To reduce contention on Oracle databases, Oracle recommends rebuilding the primary key index into a reverse key index when I/O multithreading is used. If you use and then disable I/O multithreading, Oracle recommends rebuilding the primary key index as a non-reverse index. For more information on reverse key indexes, see Indexes and Index-Organized Tables in *Oracle Database Concepts*.

Use the following basic steps to rebuild the Store table index for Oracle database:

1. Login to the Oracle database under the Store schema name. The Store schema name may or may not be the same as the data source user name.
2. Use the PL/SQL script found in Build a Reverse Index for an Oracle Database or Build a Non-Reverse Index for an Oracle Database to rebuild the Store table index as needed. Replace <*Store Table Name*> in each script with the Store table name as described in Using Prefixes with a JDBC Store. See Execution of PL/SQL Subprograms in *Oracle Database Concepts*.

> **Note**
>
> Oracle recommends reverse indexes for I/O multithreading and non-reverse indexes for single threaded I/O.

Build a Reverse Index for an Oracle Database

To rebuild the Store table index as a reverse index for an Oracle database, run the following PL/SQL block as the store database user:

```
declare
  idx           user_ind_columns.index_name%TYPE;
  alter_stmt    VARCHAR2(200);
begin
  select index_name into idx from user_ind_columns where table_name =
 <Store Table Name> and column_name = 'ID';
  alter_stmt := 'alter index ' || idx || ' rebuild reverse';
  dbms_output.put_line(alter_stmt);
  execute immediate alter_stmt;
end;
/
```

Build a Non-Reverse Index for an Oracle Database

To rebuild a reverse Store table index as a non-reverse index for Oracle database, run the following PL/SQL block as the store database user:

```
declare
  idx           user_ind_columns.index_name%TYPE;
  alter_stmt    VARCHAR2(200);
begin
  select index_name into idx from user_ind_columns where table_name =
 <Store Table Name> and column_name = 'ID';
  alter_stmt := 'alter index ' || idx || ' rebuild noreverse';
  dbms_output.put_line(alter_stmt);
  execute immediate alter_stmt;
end;
/
```

Reducing Contention in a Non-Oracle Database

For non-Oracle databases, refer to the database provider's documentation on how to reduce the contention.

6

Managing the WebLogic Persistent Store

This chapter explains how to use the administration utility and secure store data.

Administering a Persistent Store

The WebLogic Store administration utility enables administrators to troubleshoot a WebLogic persistent store. The store utility operates only on a store that is not currently opened by a running server instance. This utility can be run from a Java command line or from WebLogic Scripting Tool (WLST), as described in Store Administration Using a Java Command Line and Store Administration Using WLST.

The most common uses-cases for store administration are for compacting a file store to reduce its size and for dumping the contents of a file store of JDBC store to an XML file for troubleshooting purposes. Examples of these use cases are provided later in this section.

Table 6-1 defines the available store administration commands for Java and WLST.

Table 6-1 Persistent Store Administration Options

Java Command	WLST Method	What It Does
`help`	`helpstore`	Displays available commands, usage, and examples.
`compact`	`compactstore`	Compacts and defragments the space occupied by a file store. This command only works offline and does not work for JDBC stores. **Note:** Compacting a file store is usually not necessary if you know that file store will likely grow to the current size again. File stores automatically re-use space freed by deleted records and expand only when there is insufficient internal space for new records. Also, file stores do not normally become fragmented as most persistent records are short-lived.
`openfile`	`openfilestore`	Opens an existing file store for further operations. If a file store does not exist, a new one is created in an open state using the `-create` parameter.
`openjdbc`	`openjdbcstore`	Opens an existing JDBC store for further operations. If a JDBC store does not exist, a new one is created in an open state.
`dump`	`dumpstore`	Dumps store or connection contents in a human-readable format to user-specified XML file. The XML file format is the same format used by the diagnostic image of the persistent store.
`list`	`liststore`	Lists store names, open stores, or connections in a store.
`n/a`	`getstoreconns`	Returns a list of connections in the specified store (for script access).
`n/a`	`getopenstores`	Returns a list of opened stores (for script access).

Table 6-1 (Cont.) Persistent Store Administration Options

Java Command	WLST Method	What It Does
`opts`	`n/a`	Lists invocation options for the store administration tool.
`verbose`	`n/a`	Controls display of additional information, such as stack traces.
`close`	`closestore`	Closes a previously opened store.
`quit`	`exit`	Ends the store administration session.

A persistent store can be backed by the file system (file store) or by a JDBC-capable database (JDBC store). Except for the `openfile`/`openfilestore()` and `openjdbc`/`openjdbcstore()` options, there is no difference in the options to operate on these two different types of stores.

Most commands and methods work in terms of store names, while others also work in terms of connection names. Store connections are logical groups of records within persistent stores. For example, the JMS and JTA subsystems persist their respective records in different connections in the same file store.

Store Administration Using a Java Command Line

To open the persistent store administration utility from a Java command line, type the following:

```
> java weblogic.store.Admin
> storeadmin->
```

Accessing Store Administration Help

Type `help` for detailed descriptions on available store administration commands, as well as examples of typical command usage. For example, the following comprehensive help is provided for the `list` command, which lists store names, open stores, or connections in a store.

```
storeadmin->help list
  Command:
    list
  Description:
    lists store names, open stores, or connections in a store
  Usage:
    list [-store storename|-dir dir]
  Examples:
    list #lists all opened stores by storename
    list -store store1 #lists all connections in store1
    list -dir dir1 #lists all storenames found in dir1
```

Dumping the Contents of a File Store

Here's an example of using a series of store administration commands to ultimately export the contents of a file store named `myfilestore` into a human-readable XML file format in a temporary directory. This does not include store connection names or the actual record contents, which require the optional `-conn` and `-deep` parameters.

```
> storeadmin-> list -dir .
> storeadmin-> openfile -store myfilestore -dir .
```

```
> storeadmin-> dump -store myfilestore -out d:\tmp\filestore1-out
> storeadmin-> close -store myfilestore
```

The `list` command shows all the store names in the current directory. The `openfile` and `openjdbc` commands must be used to open and/or create a file or JDBC store first before calling certain administration functions, like `dump` and `list` (only when listing open stores). After administering an open store, you must close it using the `close` command. `WritePolicy` in generated dump is the policy that the dump tool itself uses when it opens the store.

Compacting a File Store

Here is an example of using the `compact` command to compact the space occupied by a file store in the `mystores` directory.

```
> storeadmin->compact -dir c:\mystores -tempdir c:\tmp
```

Since the compact command can only be used on an unopened file store, none of the stores that have files in the source `-dir` directory should be open. Also, the temporary `-tempdir` directory should have at least enough extra space as the source directory and should also not be under the source directory. When compact successfully completes, the newly compacted store files will be in the `mystores` directory. In addition, a new, uniquely-named directory will be created under `tmp` containing the original uncompacted store files.

Store Administration Using WLST

The WLST interface has additional methods (compared to the Java command line) such as `getopenstores` and `getstoreconns`, that return relevant Java objects and can be used for scripting in WLST.

> **Note**
>
> In this release, `ThreeStepThreshold`, `Worker Count`, and `Worker Preferred Batch Size` are not supported when using the WebLogic Scripting Tool (WLST) offline.

Accessing Store Administration Help

To access the persistent store administration utility from WLST, type the following command:

```
> java weblogic.WLST
```

Type `helpstore()` for detailed descriptions on available store administration commands, as well as examples of typical command usage. For example, the following help is provided for the `list` command, which lists store names, open stores, or connections in a store.

```
> wls:/offline> helpstore(liststore)
  lists storenames, opened stores, or connections (for interactive access)
  Parameters store and dir cannot both be specified concurrently.

  Usage: liststore(store='null',dir='null')

  @param store [optional] a previously opened JDBC or File store's name.
      If store is specified, all connections in the store are listed.
  @param dir [optional] directory for which to list available store names
      If dir is specified, all store names in the directory are listed.
```

```
  If neither store nor dir are specified, all open store names are listed.
  @return 1 on success, 0 on failure
```

Note that the parameters with an equal sign "=" are optional. For example, the `compactstore` method can be invoked as either `compactstore(dir='storename', tempdir='/tmp')` or `compactstore(store='storename')`, where `tempdir` takes the default value. Default values for optional parameters are listed in the command-specific help.

Dumping the Contents of a JDBC Store Using WLST

Here is an example of using the `dumpstore` method (`store, outfile, conn='null', deep='false'`) to export the contents of a JDBC store named `myJDBCStore` in a human-readable XML file format out to a file named `mystoredump-out.xml`. This does not include store connection names or the actual record contents, which require the optional `conn` and `deep` parameters.

```
> wls:/offline>
  openjdbcstore('myJDBCStore', 'oracle.jdbc.OracleDriver',
  'jdbc:oracle:thin:@test2k31:1521:test120a', './wlstoreadmin-dump.props',
  'jmstest', 'jmstest', '', 'jdbcstoreprefix')
  dumpstore('myJDBCStore', 'mystoredump-out')
  closestore('myJDBCStore')
```

The `openjdbcstore` and `openfilestore` methods must be used to open and/or create a store first before calling certain administration functions, like `dumpstore` and `liststore` (only when listing open stores). After administering an open store, you must close it using the `closestore` method.

Compacting a File Store Using WLST

Here is an example of a WLST script that uses the `compactstore` method (`dir,tempdir='null'`) to compact the space occupied by a file store files in the `mystores` directory.

```
> wls:/offline> compactstore('c:\mystores','c:\tmpmystore.dir')
```

Since the `compactstore()` method can only be used on unopened file stores, none of the stores that have files in the source `'dir'` directory should be open. Also, the temporary `'tempdir'` directory should have at least enough extra space as the source directory and should also not be under the source directory. When compact successfully completes, the newly compacted store files will be in the `mystores` directory. In addition, a new, uniquely-named directory will be created under `tmpmystore` containing the original uncompacted store files.

Secure File Store Data

In order to properly secure file store data, you must set appropriate directory permissions on all your file store directories.

If you require data encryption, you must use appropriate third-party encryption software.

7

Monitoring the WebLogic Persistent Store

This chapter explains how to monitor the WebLogic Server persistent store.

Monitoring a Persistent Store

You can monitor statistics for each existing persistent store and for each open store connection.

Monitoring Stores

Each persistent store is represented at run time by an instance of the PersistentStoreRuntimeMBean.

The PersistentStoreRuntimeMBean provides the following options.

Table 7-1 Persistent Store Run-time Options

Option	What It Does
CreateCount	Number of create requests issued to this persistent store.
ReadCount	Number of read requests issued to this persistent store.
UpdateCount	Number of update requests issued by this persistent store.
DeleteCount	Number of delete requests issued by this persistent store.
ObjectCount	Number of objects contained in the persistent store.
Connections	Number of active connections in the persistent store.
PhysicalWriteCount	Number of times the persistent store flushes its data to durable storage.

Monitoring Store Connections

For each open persistent store connection, the persistent store also registers a PersistentStoreConnectionRuntimemMBean.

The PersistentStoreConnectionRuntimemMBean provides the following options.

Table 7-2 Persistent Store Connection Runtime Options

Option	What It Does
CreateCount	Number of create requests issued to this connection.
ReadCount	Number of read requests issued to this connection.
UpdateCount	Number of update requests issued by this connection.
DeleteCount	Number of delete requests issued by this connection.
ObjectCount	Number of objects contained in the connection.

Table 7-3 defines most of the run-time prefix names of the WebLogic services and subsystems that can create a connection to the persistent store.

Table 7-3 Persistent Store Run-Time Prefix Names

Subsystem/Service	Run-Time Prefix Name
Deployment	`weblogic.deploy.`*`internal`* where *`internal`* is the name of the deployment connection
Diagnostic Service	`weblogic.diagnostics.`*`internal`* where *`internal`* is the logical name of the diagnostic archive connection
EJB Timer Services	`weblogic.ejb.timer.`*`internal`* where *`internal`* uniquely identifies EJB deployments in a server instance
JMS Service	JMS server: `weblogic.messaging.jmsServer.`*`internal`* where *`internal`* is the name of the JMS server connection JMS durable subscriber: `weblogic.messaging.jmsServer.durablesubs.`*`internal`* where *`internal`* is the name of the durable subscriber connection
JTA Transaction Log (TLOG)	`weblogic.transaction.`*`internal`* where *`internal`* is the name of the TLOG connection
Path Service	`weblogic.messaging.PathService.`*`internal`* where *`internal`* is the name of the path service connection

Table 7-3 (Cont.) Persistent Store Run-Time Prefix Names

Subsystem/Service	Run-Time Prefix Name
SAF Service	SAF agent `weblogic.messaging.SAFAgent@server1.`*`internal`* where *`internal`* is the name of the SAF agent's connection SAF durable subscriber: `weblogic.messaging.SAFAgent@server1.durablesubs.`*`internal`* where *`internal`* is the name of the durable subscriber connection
Web Services	`weblogic.wsee.server.store.`*`internal`* where *`internal`* is the name of the Web Service's connection