8
Data Sources and URLs

This chapter discusses connecting applications to databases using Java Database Connectivity (JDBC) data sources, as well as the URLs that describe databases. This chapter contains the following sections:

8.1 About Data Sources

Data sources are standard, general-use objects for specifying databases or other resources to use. The JDBC 2.0 extension application programming interface (API) introduced the concept of data sources. For convenience and portability, data sources can be bound to Java Naming and Directory Interface (JNDI) entities, so that you can access databases by logical names.

The data source facility provides a complete replacement for the previous JDBC `DriverManager` facility. You can use both facilities in the same application, but it is recommended that you transition your application to data sources.

This section covers the following topics:

8.1.1 Overview of Oracle Data Source Support for JNDI

The JNDI standard provides a way for applications to find and access remote services and resources. These services can be any enterprise services. However, for a JDBC application, these services would include database connections and services.

JNDI enables an application to use logical names in accessing these services, removing vendor-specific syntax from application code. JNDI has the functionality to associate a logical name with a particular source for a desired service.

All Oracle JDBC data sources are JNDI-referenceable. The developer is not required to use this functionality, but accessing databases through JNDI logical names makes the code more portable.

> **Note:**
>
> Using JNDI functionality requires the `jndi.jar` file to be in the `CLASSPATH` environment variable. This file is included with the Java products on the installation CD. You must add it to the `CLASSPATH` environment variable separately.

8.1.2 Features and Properties of Data Sources

By using the data source functionality with JNDI, you do not need to register the vendor-specific JDBC driver class name and you can use logical names for URLs and other properties. This ensures that the code for opening database connections is portable to other environments.

The DataSource Interface and Oracle Implementation

A JDBC data source is an instance of a class that implements the standard `javax.sql.DataSource` interface:

```
public interface DataSource
{
   Connection getConnection() throws SQLException;
   Connection getConnection(String username, String password)
      throws SQLException;
   ...
}
```

Oracle implements this interface with the `OracleDataSource` class in the `oracle.jdbc.pool` package. The overloaded `getConnection` method returns a connection to the database.

To use other values, you can set properties using appropriate setter methods. For alternative user names and passwords, you can also use the `getConnection` method that takes these parameters as input. This would take priority over the property settings.

> **Note:**
>
> The `OracleDataSource` class and all subclasses implement the `java.io.Serializable` and `javax.naming.Referenceable` interfaces.

Properties of DataSource

`The OracleDataSource` class, as with any class that implements the `DataSource` interface, provides a set of properties that can be used to specify a database to connect to. These properties follow the JavaBeans design pattern.

The following tables list the `OracleDataSource` standard properties and Oracle extensions respectively.

> **Note:**
>
> Oracle does not implement the standard `roleName` property.

Table 8-1 Standard Data Source Properties

Name	Type	Description
`databaseName`	`String`	Name of the particular database on the server.
`dataSourceName`	`String`	Name of the underlying data source class. For connection pooling, this is an underlying pooled connection data source class. For distributed transactions, this is an underlying XA data source class.
`description`	`String`	Description of the data source.
`networkProtocol`	`String`	Network protocol for communicating with the server. For Oracle, this applies only to the JDBC Oracle Call Interface (OCI) drivers and defaults to `tcp`.
`password`	`String`	Password for the connecting user.
`portNumber`	`int`	Number of the port where the server listens for requests
`serverName`	`String`	Name of the database server
`user`	`String`	User name to log in

Note:

For security reasons, there is no `getPassword()` method.

Table 8-2 Oracle Extended Data Source Properties

Name	Type	Description
`connectionCacheName`	`String`	Specifies the name of the cache. This cannot be changed after the cache has been created.
`connectionCacheProperties`	`java.util.Properties`	Specifies properties for implicit connection cache.
`connectionCachingEnabled`	`Boolean`	Specifies whether implicit connection cache is in use.
`connectionProperties`	`java.util.Properties`	Specifies the connection properties.
`driverType`	`String`	Specifies Oracle JDBC driver type. It can be one of `oci`, `thin`, or `kprb`.
`fastConnectionFailoverEnabled`	`Boolean`	Specifies whether Fast Connection Failover is in use.
`implicitCachingEnabled`	`Boolean`	Specifies whether the implicit statement connection cache is enabled.
`loginTimeout`	`int`	Specifies the maximum time in seconds that this data source will wait while attempting to connect to a database.
`logWriter`	`java.io.PrintWriter`	Specifies the log writer for this data source.
`maxStatements`	`int`	Specifies the maximum number of statements in the application cache.

Table 8-2 (Cont.) Oracle Extended Data Source Properties

Name	Type	Description
serviceName	String	Specifies the database service name for this data source.
tnsEntry	String	Specifies the TNS entry name. The TNS entry name corresponds to the TNS entry specified in the tnsnames.ora configuration file. Enable this OracleXADataSource property when using the Native XA feature with the OCI driver, to access Oracle pre-8.1.6 databases and later. If the tnsEntry property is not set when using the Native XA feature, then a SQLException with error code ORA-17207 is thrown
url	String	Specifies the URL of the database connection string. Provided as a convenience, it can help you migrate from an older Oracle Database. You can use this property in place of the Oracle tnsEntry and driverType properties and the standard portNumber, networkProtocol, serverName, and databaseName properties.
nativeXA	Boolean	Allows an OracleXADataSource using the Native XA feature with the OCI driver, to access Oracle pre-8.1.6 databases and later. If the nativeXA property is enabled, be sure to set the tnsEntry property as well. This property is only for OracleXADatasource. This DataSource property defaults to false.
ONSConfiguration	String	Specifies the ONS configuration string that is used to remotely subscribe to FAN/ONS events.

Note:

- This table omits properties that supported the deprecated connection cache based on OracleConnectionCache.
- Because Native XA performs better than Java XA, use Native XA whenever possible.

Use the setConnectionProperties method to set the properties of the connection and the setConnectionCacheProperties method to set the properties of the connection cache.

If you are using the server-side internal driver, that is, the driverType property is set to kprb, then any other property settings are ignored.

If you are using the JDBC Thin or OCI driver, then note the following:

- A URL setting can include settings for user and password, as in the following example, in which case this takes precedence over individual user and password property settings:

```
jdbc:oracle:thin:HR/<password>@localhost:5221:orcl
```

- Settings for `user` and `password` are required, either directly through the URL setting or through the `getConnection` call. The `user` and `password` settings in a `getConnection` call take precedence over any property settings.
- If the `url` property is set, then any `tnsEntry`, `driverType`, `portNumber`, `networkProtocol`, `serverName`, and `databaseName` property settings are ignored.
- If the `tnsEntry` property is set, which presumes the `url` property is not set, then any `databaseName`, `serverName`, `portNumber`, and `networkProtocol` settings are ignored.
- If you are using an OCI driver, which presumes the `driverType` property is set to `oci`, and the `networkProtocol` is set to `ipc`, then any other property settings are ignored.

Also, note that `getConnectionCacheName()` will return the name of the cache only if the `ConnectionCacheName` property of the data source is set after caching is enabled on the data source.

8.1.3 Creating a Data Source Instance and Connecting

This section shows an example of the most basic use of a data source to connect to a database, without using JNDI functionality. Note that this requires vendor-specific, hard-coded property settings.

Create an `OracleDataSource` instance, initialize its connection properties as appropriate, and get a connection instance, as in the following example:

```
OracleDataSource ods = new OracleDataSource();
ods.setDriverType("oci");
ods.setServerName("localhost");
ods.setNetworkProtocol("tcp");
ods.setDatabaseName(<database_name>);
ods.setPortNumber(5221);
ods.setUser("HR");
ods.setPassword("hr");
Connection conn = ods.getConnection();
```

Or, optionally, override the user name and password, as follows:

```
Connection conn = ods.getConnection("OE", "oe");
```

8.1.4 Creating a Data Source Instance, Registering with JNDI, and Connecting

This section exhibits JNDI functionality in using data sources to connect to a database. Vendor-specific, hard-coded property settings are required only in the portion of code that binds a data source instance to a JNDI logical name. From that point onward, you can create portable code by using the logical name in creating data sources from which you will get your connection instances.

> **Note:**
>
> Creating and registering data sources is typically handled by a JNDI administrator, not in a JDBC application.

Initialize Data Source Properties

Create an `OracleDataSource` instance, and then initialize its properties as appropriate, as in the following example:

```
OracleDataSource ods = new OracleDataSource();
ods.setDriverType("oci");
ods.setServerName("localhost");
ods.setNetworkProtocol("tcp");
ods.setDatabaseName("816");
ods.setPortNumber(5221);
ods.setUser("HR");
ods.setPassword("hr");
```

Register the Data Source

Once you have initialized the connection properties of the `OracleDataSource` instance `ods`, as shown in the preceding example, you can register this data source instance with JNDI, as in the following example:

```
Context ctx = new InitialContext();
ctx.bind("jdbc/sampledb", ods);
```

Calling the JNDI `InitialContext()` constructor creates a Java object that references the initial JNDI naming context. System properties, which are not shown, instruct JNDI which service provider to use.

The `ctx.bind` call binds the `OracleDataSource` instance to a logical JNDI name. This means that anytime after the `ctx.bind` call, you can use the logical name `jdbc/sampledb` in opening a connection to the database described by the properties of the `OracleDataSource` instance `ods`. The logical name `jdbc/sampledb` is logically bound to this database.

The JNDI namespace has a hierarchy similar to that of a file system. In this example, the JNDI name specifies the subcontext `jdbc` under the root naming context and specifies the logical name `sampledb` within the `jdbc` subcontext.

The `Context` interface and `InitialContext` class are in the standard `javax.naming` package.

> **Note:**
>
> The JDBC 2.0 Specification requires that all JDBC data sources be registered in the `jdbc` naming subcontext of a JNDI namespace or in a child subcontext of the `jdbc` subcontext.

Open a Connection

To perform a lookup and open a connection to the database logically bound to the JNDI name, use the logical JNDI name. Doing this requires casting the lookup result, which is otherwise a Java `Object`, to `OracleDataSource` and then using its `getConnection` method to open the connection.

Here is an example:

```
OracleDataSource odsconn = (OracleDataSource)ctx.lookup("jdbc/sampledb");
Connection conn = odsconn.getConnection();
```

8.1.5 Supported Connection Properties

For a detailed list of connection properties that Oracle JDBC drivers support, see the *Oracle Database JDBC Java API Reference*.

> **Note:**
>
> The JDBC Thin Driver connection property `oracle.jdbc.ReadTimeout` and the JDBC method `java.sql.Connection.setNetworkTimeout` may not behave as expected, when Dead Connection Detection (DCD) is enabled in the old way in Oracle Database. See the following My Oracle Support Note for more information about the new implementation of DCD: https://support.oracle.com/rs?type=doc&id=1591874.1

8.1.6 About Using Roles for SYS Login

To specify the role for the `SYS` login, use the `internal_logon` connection property. To log on as `SYS`, set the `internal_logon` connection property to `SYSDBA` or `SYSOPER`.

> **Note:**
>
> The ability to specify a role is supported only for the `sys` user name.

For a bequeath connection, we can get a connection as `SYS` by setting the `internal_logon` property. For a remote connection, we need additional password file setting procedures.

8.1.7 Configuring Database Remote Login

Before the JDBC Thin driver can connect to the database as `SYSDBA`, you must configure the user, because Oracle Database security system requires a password file for remote connections as an administrator. Perform the following:

1. Set a password file on the server-side or on the remote database, using the `orapwd` password utility. You can add a password file for user `SYS` as follows:

   - In UNIX

     ```
     orapwd file=$ORACLE_HOME/dbs/orapwORACLE_SID entries=200
     Enter password: password
     ```

   - In Microsoft Windows

     ```
     orapwd file=%ORACLE_HOME%\database\PWDORACLE_SID.ora entries=200
     Enter password: password
     ```

   In this case, `file` is the name of the password file, *`password`* is the password for user `SYS`. It can be altered using the `ALTER USER` statement in SQL Plus. You should set `entries` to a value higher than the number of entries you expect.

   The syntax for the password file name is different on Microsoft Windows and UNIX.

> **See Also:**
>
> *Oracle Database Administrator's Guide*

2. Enable remote login as `SYSDBA`. This step grants `SYSDBA` and `SYSOPER` system privileges to individual users and lets them connect as themselves.

   Stop the database, and add the following line to `init`*`service_name`*`.ora`, in UNIX, or `init.ora`, in Microsoft Windows:

   ```
   remote_login_passwordfile=exclusive
   ```

   The `init`*`service_name`*`.ora` file is located at *`ORACLE_HOME`*`/dbs/` and also at *`ORACLE_HOME`*`/admin/db_name/pfile/`. Ensure that you keep the two files synchronized.

   The `init.ora` file is located at `%ORACLE_BASE%\ADMIN\db_name\pfile\`.

3. Change the password for the `SYS` user. This is an optional step.

   ```
   PASSWORD sys
         Changing password for sys
   New password: password
   Retype new password: password
   ```

4. Verify whether `SYS` has the `SYSDBA` privilege.

   ```
   SQL> select * from v$pwfile_users;
   USERNAME                       SYSDB        SYSOP
   ----------------------         ---------    ---------
   SYS                            TRUE         TRUE
   ```

5. Restart the remote database.

Example 8-1 Using SYS Login To Make a Remote Connection

```
//This example works regardless of language settings of the database.
 /** case of remote connection using sys **/
import java.sql.*;
import oracle.jdbc.*;
import oracle.jdbc.pool.*;
// create an OracleDataSource
OracleDataSource ods = new OracleDataSource();
// set connection properties
java.util.Properties prop = new java.util.Properties();
prop.put("user", "sys");
prop.put("password", "sys");
prop.put("internal_logon", "sysoper");
ods.setConnectionProperties(prop);
// set the url
// the url can use oci driver as well as:
// url = "jdbc:oracle:oci8:@remotehost"; the remotehost is a remote database
String url = "jdbc:oracle:thin:@localhost:5221/orcl";
ods.setURL(url);
// get the connection
Connection conn = ods.getConnection();
...
```

8.1.8 Using Bequeath Connection and SYS Logon

The following example illustrates how to use the `internal_logon` and `SYSDBA` arguments to specify the `SYS` login. This example works regardless of the database's national-language settings of the database.

```
/** Example of bequeath connection **/
import java.sql.*;
import oracle.jdbc.*;
import oracle.jdbc.pool.*;

// create an OracleDataSource instance
OracleDataSource ods = new OracleDataSource();

// set neccessary properties
java.util.Properties prop = new java.util.Properties();
prop.put("user", "sys");
prop.put("password", "sys");
prop.put("internal_logon", "sysdba");
ods.setConnectionProperties(prop);

// the url for bequeath connection
String url = "jdbc:oracle:oci8:@";
ods.setURL(url);

// retrieve the connection
Connection conn = ods.getConnection();
...
```

8.1.9 Setting Properties for Oracle Performance Extensions

Some of the connection properties are for use with Oracle performance extensions. Setting these properties is equivalent to using corresponding methods on the `OracleConnection` object, as follows:

- Setting the `defaultRowPrefetch` property is equivalent to calling `setDefaultRowPrefetch`.
- Setting the `remarksReporting` property is equivalent to calling `setRemarksReporting`.

> **See Also:**
>
> "About Reporting DatabaseMetaData TABLE_REMARKS"

Example

The following example shows how to use the `put` method of the `java.util.Properties` class, in this case, to set Oracle performance extension parameters.

```
//import packages and register the driver
import java.sql.*;
import java.math.*;
import oracle.jdbc.*;
import oracle.jdbc.pool.OracleDataSource;

//specify the properties object
java.util.Properties info = new java.util.Properties();
```

```
info.put ("user", "HR");
info.put ("password", "hr");
info.put ("defaultRowPrefetch","20");
info.put ("defaultBatchValue", "5");

//specify the datasource object
OracleDataSource ods = new OracleDataSource();
ods.setURL("jdbc:oracle:thin:@localhost:5221/orcl");
ods.setUser("HR");
ods.setPassword("hr");
ods.setConnectionProperties(info);
...
```

8.1.10 Support for Network Data Compression

Starting from Oracle Database 12*c* Release 2 (12.2.0.1), the JDBC Thin driver supports network data compression. Network data compression reduces the size of the session data unit (SDU) transmitted over a data connection and reduces the time required to transmit a SQL query and the result across the network. The benefits are more significant in case of Wireless Area Network (WAN). For enabling network data compression, you must set the connection properties in the following way:

> **Note:**
>
> Network compression does not work for streamed data.

```
...
OracleDataSource ds = new OracleDataSource();
Properties prop = new Properties();
prop.setProperty("user","user1");
prop.setProperty("password",<password>);

// Enabling Network Compression
prop.setProperty("oracle.net.networkCompression","on");

//Optional configuration for setting the client compression threshold.
prop.setProperty("oracle.net.networkCompressionThreshold","1024");

ds.setConnectionProperties(prop);
ds.setURL(url);
Connection conn = ds.getConnection();
...
```

8.2 Database URLs and Database Specifiers

Database URLs are strings. The complete URL syntax is:

```
jdbc:oracle:driver_type:[username/password]@database_specifier
```

> **Note:**
>
> - The brackets indicate that the *username/password* pair is optional.
> - kprb, the internal server-side driver, uses an implicit connection. Database URLs for the server-side driver end after the *driver_type*.

The first part of the URL specifies which JDBC driver is to be used. The supported *driver_type* values are thin, oci, and kprb.

The remainder of the URL contains an optional user name and password separated by a slash, an @, and the database specifier, which uniquely identifies the database to which the application is connected. Some database specifiers are valid only for the JDBC Thin driver, some only for the JDBC OCI driver, and some for both.

8.2.1 Support for Longer Passwords

In previous releases, the Oracle Database password length was up to 30 bytes. Starting with Oracle Database Release 23ai, Oracle Database supports passwords up to 1024 bytes in length.

The increased maximum password length provides the following benefits:

- It accommodates passwords that are used by Oracle Identity Cloud Service (IDCS) and Identity Access Management (IAM). The increase to 1024 bytes enables uniform password rules for all Cloud deployments.
- The 30-byte limitation was too restrictive when password multibyte characters used more than 1 byte in an NLS configuration.

8.2.2 Support for Internet Protocol Version 6

This release of Oracle JDBC drivers supports Internet Protocol Version 6 (IPv6) addresses in the JDBC URL and machine names that resolve to IPv6 addresses. IPv6 is a new Network layer protocol designed by the Internet Engineering Task Force (IETF) to replace the current version of Internet Protocol, Internet Protocol Version 4 (IPv4). The primary benefit of IPv6 is a large address space, derived from the use of 128-bit addresses. IPv6 also improves upon IPv4 in areas such as routing, network auto configuration, security, quality of service, and so on.

> **Note:**
>
> - An IPv6 Client can support only IPv6 Servers or servers with dual protocol support, that is, support for both IPv6 and IPv4 protocols. Conversely, an IPv6 Server can support only IPv6 clients or dual protocol clients.
> - IPv6 is supported only with single instance Database servers and not with Oracle RAC.

If you want to use a literal IPv6 address in a URL, then you should enclose the literal address enclosed in a left bracket ([) and a right bracket (]). For example: [2001:0db8:7654:3210:FEDC:BA98:7654:3210]. So, a JDBC URL, using an IPv6 address will look like the following:

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)
  (HOST=[2001:0db8:7654:3210:FEDC:BA98:7654:3210])(PORT=5521))
  (CONNECT_DATA=(SERVICE_NAME=sales.example.com))
```

> **Note:**
>
> All the new System classes that are required for IPv6 support are loaded when Java is enabled during database initialization. So, if your application does not have any IPv6 addressing, then you do not need to change your code to use IPv6 functionality. However, if your application has either IPv6 only or both IPv6 and IPv4 addressing, then you should set the `java.net.preferIPv6Addresses` system property in the command line. This enables the Oracle JVM to load appropriate libraries. These libraries are loaded once and cannot be reloaded without restarting the Java process.

8.2.3 Support for HTTPS Proxy Configuration

Oracle Database Release 18c JDBC drivers support HTTPS Proxy Configuration. HTTPS Proxy enables tunnelling secure connections over forward HTTP proxy using the `HTTP CONNECT` method. This helps in accessing the public cloud database service as it eliminates the requirement to open an outbound port on a client side firewall. This parameter is applicable only to the connect descriptors where `PROTOCOL=TCPS`. This is similar to the web browser setting for intranet users who want to connect to internet hosts.

For configuring HTTPS Proxy, add details to the `ADDRESS` part of the Connection String as shown in the following code snippet:

```
(DESCRIPTION=

  (ADDRESS=(HTTPS_PROXY=sales-proxy)(HTTPS_PROXY_PORT=8080)(PROTOCOL=TCPS)
(HOST=sales2-svr)(PORT=443))

  (CONNECT_DATA=(SERVICE_NAME=sales.us.example.com)))
```

8.2.4 Database Specifiers

Table 8-3, shows the possible database specifiers, listing which JDBC drivers support each specifier.

Table 8-3 Supported Database Specifiers

Specifier	Supported Drivers	Example

Oracle Net connect descriptor	Thin, OCI	Thin, using an address list:
		
		`jdbc:oracle:thin:@(DESCRIPTION=`
		`  (LOAD_BALANCE=on)`
		`(ADDRESS_LIST=`
		`  (ADDRESS=(PROTOCOL=TCP)(HOST=host1)  (PORT=5221))`
		`(ADDRESS=(PROTOCOL=TCP)(HOST=host2)(PORT=5221)))`
		`(CONNECT_DATA=(SERVICE_NAME=orcl)))`
		
		OCI, using a cluster:
		
		`jdbc:oracle:oci:@(DESCRIPTION=`
		`  (ADDRESS=(PROTOCOL=TCP)(HOST=cluster_alias)`
		`  (PORT=5221))`
		`  (CONNECT_DATA=(SERVICE_NAME=orcl)))`

Thin-style service name	Thin	This is deprecated in favor of the Easy Connect Plus syntax. Refer to "Support for Easy Connect Plus" for more details. `jdbc:oracle:thin:HR/<password>@localhost:5221/orcl`

LDAP syntax	Thin	This is deprecated in favor of the Easy Connect Plus syntax. Refer to "Support for Easy Connect Plus" for more details.
		
		`jdbc:oracle:thin:@ldap:ldap.example.com:7777/`
		`sales,cn=OracleContext,dc=com`

Easy Connect Plus syntax	Thin	This format can be used to connect using the following:
		• The TLS protocol:
		
		`jdbc:oracle:thin:@tcps://salesserver1:1521/`
		`sales.us.example.com?wallet_location=/work/`
		`wallet/`
		
		• Plain TCP:
		
		`jdbc:oracle:thin:@//salesserver1:1521/`
		`sales.us.example.com`
		
		• Net connect descriptor stored in an LDAP directory:
		
		`jdbc:oracle:thin:@ldaps://myldapserver:1636/`
		`cn=orcl,cn=OracleContext,dc=example,dc=com`
		
		Refer to the "oracle.jdbc.OracleDriver class for more details

Table 8-3 (Cont.) Supported Database Specifiers

Specifier	Supported Drivers	Example

Bequeath connection	Thin, OCI	For the Thin driver:
		`jdbc:oracle:thin:@(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq))(ENVS=ORACLE_HOME=/var/lib/oracle/dbhome,ORACLE_SID=oraclesid))`
		Refer to "Support for the Bequeath Protocol" for more details.
		For the OCI driver, it is empty, that is, nothing is specified after @
		`jdbc:oracle:oci:HR/<password>/@`

TNSNames alias	Thin, OCI	Refer to "TNSNames Alias Syntax" for details.
		`jdbc:oracle:thin:@<alias-name>?TNS_ADMIN=/work/tnsadmin/`
		You can also configure the TNSNames alias through an API:
		`OracleDataSource ods = new OracleDataSource();`
		`ods.setTNSEntryName("MyTNSAlias");`

Configuration Provider	Thin	`jdbc:oracle:thin:@config-<provider-type>://<parameters>` For example: • `jdbc:oracle:thin:@config-file://config.json` • `jdbc:oracle:thin:@config-https://myserver/config/myapp?key=dev` Refer to "the oracle.jdbc.OracleDriver Class and the oracle.jdbc.spi.OracleConfigurationProvider Interface for more details

8.2.5 Thin-style Service Name Syntax

Thin-style service names are supported only by the JDBC Thin driver. The syntax is:

```
@host_name:port_number/service_name
```

For example:

```
jdbc:oracle:thin:HR/<password>@localhost:5221/orcl
```

> **Note:**
>
> The JDBC Thin driver supports only the TCP/IP protocol.

8.2.6 Support for Easy Connect Plus

Oracle Database Release 19c JDBC driver introduced enhancements to the EasyConnect URL with many capabilities and this new enhanced Easy Connection URL is called Easy Connect Plus.

The EasyConnect URL supported simplified connection strings and the TCP transfer protocol. The goal of the enhancement to the EasyConnect URL is to simplify the client-side deployment and configuration, while connecting to the database. This enhancement can eliminate the need of using the `tnsnames.ora` file and you do not need to specify the `TNS_ADMIN` environment variable. The main features of Easy Connect Plus are the following:

- Support for TCPS Protocol
- Support for Multiple Hosts and Ports
- Support to Pass Connection Properties in the Connection String

8.2.6.1 Support for TCPS Protocol

The EasyConnect URL supported only the TCP transport protocol. However, the Easy Connect Plus supports both TCP and TCPS protocols. This enhancement simplifies the client configurations to Oracle Database Cloud Services that mandate TCPS connections.

For example, if you have a EasyConnect URL in the following format:

```
jdbc:oracle:thin:@(DESCRIPTION=
(ADDRESS=(PROTOCOL=tcps)(HOST=salesserver1)(PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=sales.us.example.com)))
```

Then, the Easy Connect Plus URL will be:

```
jdbc:oracle:thin:@tcps:salesserver1:1521/sales.us.example.com
```

8.2.6.2 Support for LDAP and LDAPS

Starting from Oracle Database Release 23ai, the Easy Connect Plus URL supports LDAP and LDAPS protocols.

Syntax

The syntax follows a combination of the LDAP URL syntax and the Easy Connect Plus syntax:

```
ldap[s]://host[:port]/name[,context]?[parameter=value{&parameter=value}]
```

For example:

```
sqlplus "<user_name>/<password>@ldaps://<host_name>/test?
DIRECTORY_SERVER_TYPE=oid&WALLET_LOCATION=/oracle/network/admin
&AUTHENTICATE_BIND=true&AUTHENTICATE_BIND_METHOD=LDAPS_SIMPLE_AUTH"
```

Parameters

The following table discusses the parameters used in the syntax:

Parameter Name	Description	Default Value
`host`	Specifies the host name. It is a compulsory parameter.	Not applicable
`name`	Specifies the alias name to be resolved. It is a compulsory parameter.	Not applicable
`port`	Specifies the port to be used. It is an optional parameter.	389 for LDAP and 636 for LDAPS
`context`	Specifies the context to be used. It is an optional parameter.	`cn=OracleContext`
`directory_server_type`	Specifies the directory server type to be used. The supported values are Oracle Internet Directory (OID), Oracle Unified Directory (OUD), and Active Directory (AD). It is an optional parameter.	OID
`wallet_location`	Specifies the wallet location to be used. It is an optional parameter.	Not applicable
`authenticate_bind`	Specifies whether LDAP adapter should use a wallet for authentication or not. Can be `true` or `false`. It is an optional parameter.	`false`
`authenticate_bind_method`	Specifies the authentication method to be used. The values can be either `LDAPS_SIMPLE_AUTH` or `NONE`. It is an optional parameter.	`NONE`

The `directory_server_type`, `wallet_location`, `authenticate_bind`, and `authenticate_bind_method` are position independent.

8.2.6.3 Support for Multiple Hosts and Ports

Using Easy Connect Plus, you can specify a comma-separated list of hosts.

The EasyConnect URL allowed only single host name and single port. But, Easy Connect Plus allows specifying multiple hosts and multiple ports in the connection string, which can be used to connect to a database with load balancing turned on. This enhanced Easy Connect syntax, using multiple hosts and ports, is as follows:

```
[[protocol:]//]host1{,host12}[:port1]{,host2:port2}{;host1{,host12}[:port1]}
[/[service_name][:server][/instance_name]][?
parameter_name=value{&parameter_name=value}]
```

> **See Also:**
>
> Support for Easy Connect Plus

For example, if you want to specify two hosts with a single port number in the connection using the following EasyConnect URL:

```
jdbc:oracle:thin:@(DESCRIPTION=
(ADDRESS_LIST= (LOAD_BALANCE=ON) (ADDRESS=(PROTOCOL=tcp)(HOST=salesserver1)
(PORT=1521)) (ADDRESS=(PROTOCOL=tcp)(HOST=salesserver2)(PORT=1521)))
(CONNECT_DATA=(SERVICE_NAME=sales.us.example.com)))
```

Then, the Easy Connect Plus URL will be the following:

```
jdbc:oracle:thin:@tcp:salesserver1,salesserver2:1521/sales.us.example.com
```

Again, if you want to specify multiple hosts as a list, where each list follows its own port number, using the following EasyConnect URL:

```
jdbc:oracle:thin:@(DESCRIPTION= (ADDRESS_LIST= (LOAD_BALANCE=ON)
(ADDRESS=(PROTOCOL=tcp)(HOST=salesserver1)(PORT=1521)) (ADDRESS=(PROTOCOL=tcp)
(HOST=salesserver2)(PORT=1522))
(ADDRESS=(PROTOCOL=tcp)(HOST=salesserver3)(PORT=1522)))
(CONNECT_DATA=(SERVICE_NAME=sales.us.example.com)))
```

Then, the Easy Connect Plus URL will be the following:

```
jdbc:oracle:thin:@tcp:salesserver1:1521,salesserver2,salesserver3:1522/
sales.us.example.com
```

8.2.6.4 Support to Pass Connection Properties in the Connection String

Easy Connect Plus supports specification of connection properties as name-value pairs in the connection URL. After the `?` delimiter, Easy Connect Plus supports a list of parameters as name-value pairs, which can be optionally specified.

Remember the following points while specifying the name-value pairs:

- Use a question mark sign (`?`) to indicate the start of the name-value pairs and an ampersand sign (`&`) as a delimiter between each name-value pair.
- Specify the entire connection string as a single string.
- Use the backslash (`\`) escape character if there are any special characters in the value.
- Place white spaces within double-quotes if they are required as part of the value because leading and trailing white spaces are ignored within the parameter values.

The following table lists a few of the supported parameters:

Parameter Name	Old URL Example	New URL Example
wallet_location	jdbc:oracle:thin:@(DESCRIPTION= (ADDRESS=(PROTOCOL=tcps) (HOST=salesserver1) (PORT=1521)) (CONNECT_DATA=(SERVICE_NAME=sales.us.example.com))	jdbc:oracle:thin:@tcps:salesserver1.com:1521/sales.us.example.com?wallet_location=/Users/jsmith/DBCloudService/wallet_JDBCTEST&oracle.net.ssl_server_cert_dn=\"CN=salesserver2.com.com,OU=Oracle BMCS US,O=Oracle Corporation,L=Redwood City,ST=California,C=US\"" ;
ssl_server_dn_match		
ssl_server_cert_dn	jdbc:oracle:thin:@(DESCRIPTION= (ADDRESS=(PROTOCOL=tcps) (HOST=salesserver1) (PORT=1521)) (SECURITY= (SSL_SERVER_DN_MATCH=TRUE) (SSL_SERVER_CERT_DN=cn=sales,cn=OracleContext,dc=us,dc=example,dc=com)) (CONNECT_DATA=(SERVICE_NAME=sales.us.example.com)))	jdbc:oracle:thin:@tcps:salesserver1:1521/sales.us.example.com?ssl_server_cert_dn="cn=sales,cn=OracleContext,dc=us,dc=example,dc=com"
https_proxy and https_proxy_port	jdbc:oracle:thin:@(DESCRIPTION= (ADDRESS=(PROTOCOL=tcps) (HOST=salesserver1) (PORT=1521) (https_proxy=www-proxy.mycompany.com) (https_proxy_port=80)) (CONNECT_DATA=(SERVICE_NAME=sales.us.example.com)))	jdbc:oracle:thin:@tcps:salesserver1:1521/sales.us.example.com?https_proxy=www-proxy.mycompany.com&https_proxy_port=80
connect_timeout	jdbc:oracle:thin:@(DESCRIPTION= (retry_count=3) (connect_timeout=60) (transport_connect_timeout=30) (ADDRESS=(PROTOCOL=tcp) (HOST=salesserver1) (PORT=1521)) (CONNECT_DATA=(SERVICE_NAME=sales.us.example.com)))	jdbc:oracle:thin:@tcps:salesserver1:1521/sales.us.example.com?connect_timeout=60&transport_connect_timeout=30&retry_count=3

The list of supported keywords under DESCRIPTION is as follows:

- ENABLE
- FAILOVER
- LOAD_BALANCE
- RECV_BUF_SIZE

- SEND_BUF_SIZE
- SDU
- SOURCE_ROUTE
- RETRY_COUNT
- RETRY_DELAY
- CONNECT_TIMEOUT
- TRANSPORT_CONNECT_TIMEOUT

> **Note:**
>
> - When your application uses a SOCKS proxy to connect to Oracle Database (for example, a Bastion in the Oracle Cloud), then the value of the TRANSPORT_CONNECT_TIMEOUT parameter is ignored.
> - Starting from Oracle Database Release 23ai, the default value of the TRANSPORT_CONNECT_TIMEOUT parameter is 20 seconds.

8.2.7 Support for Delay in Connection Retries

The RETRY_DELAY connection attribute specifies the delay between connection retries in seconds by default. Starting from Oracle Database Release 23ai, the default value of this parameter is 1 second.

Starting from Oracle Database Release 21c, you can also specify the delay in milliseconds and minutes, using the following units:

- ms (milliseconds)
- s (seconds)
- minutes (m)

The following code snippet shows how to use this attribute:

```
(DESCRIPTION_LIST=
  (DESCRIPTION=
   (CONNECT_TIMEOUT=10)(RETRY_COUNT=3)(RETRY_DELAY=800ms)
   (ADDRESS_LIST=
    (ADDRESS=(PROTOCOL=tcp)(HOST=myhost1)(PORT=1521))
    (ADDRESS=(PROTOCOL=tcp)(HOST=myhost2)(PORT=1521)))
   (CONNECT_DATA=(SERVICE_NAME=example1.com)))
  (DESCRIPTION=
   (CONNECT_TIMEOUT=60)(RETRY_COUNT=1)(RETRY_DELAY=5s)
   (ADDRESS_LIST=
    (ADDRESS=(PROTOCOL=tcp)(HOST=myhost3)(PORT=1521))
    (ADDRESS=(PROTOCOL=tcp)(HOST=myhost4)(PORT=1521)))
   (CONNECT_DATA=(SERVICE_NAME=example2.com))))
```

8.2.8 TNSNames Alias Syntax

You can find the available TNSNAMES entries listed in the tnsnames.ora file on the client computer from which you are connecting. On Windows, this file is located in the

ORACLE_HOME\NETWORK\ADMIN directory. On UNIX systems, you can find it in the *ORACLE_HOME* directory or the directory indicated in your TNS_ADMIN environment variable.

For example, if you want to connect to the database on host myhost as user HR with password hr that has a TNSNAMES entry of MyHostString, then write the following:

```
OracleDataSource ods = new OracleDataSource();
ods.setTNSEntryName("MyTNSAlias");
ods.setUser("HR");
ods.setPassword("hr");
ods.setDriverType("oci");
Connection conn = ods.getConnection();
```

The oracle.net.tns_admin system property must be set to the location of the tnsnames.ora file so that the JDBC Thin driver can locate the tnsnames.ora file. For example:

```
System.setProperty("oracle.net.tns_admin", "c:\\Temp");
String url = "jdbc:oracle:thin:@tns_entry";
```

> **Note:**
>
> When using TNSNames with the JDBC Thin driver, you must set the oracle.net.tns_admin property to the directory that contains your tnsnames.ora file.
>
> ```
> java -Doracle.net.tns_admin=$ORACLE_HOME/network/admin
> ```

8.2.9 LDAP Syntax

This section describes the database specifiers that you can use with LDAP servers.

An example of database specifier using the Lightweight Directory Access Protocol (LDAP) syntax is as follows:

```
"jdbc:oracle:thin:@ldap:ldap.example.com:7777/sales,cn=OracleContext,dc=com"
```

When using TLS, change this syntax to the following:

```
"jdbc:oracle:thin:@ldaps:ldap.example.com:7777/sales,cn=OracleContext,dc=com"
```

> **Note:**
>
> The JDBC Thin driver can use LDAP over TLS to communicate with Oracle Internet Directory if you substitute ldap: with ldaps: in the database specifier. The LDAP server must be configured to use TLS. If it is not, then the connection attempt will hang.

The JDBC Thin driver supports failover of a list of LDAP servers during the service name resolution process, without the need for a hardware load balancer. Also, client-side load balancing is supported for connecting to LDAP servers. A list of space separated LDAP URLs syntax is used to support failover and load balancing.

When a list of LDAP URLs is specified, both failover and load balancing are enabled by default. You can use the oracle.net.ldap_loadbalance connection property to disable load balancing, and the oracle.net.ldap_failover connection property to disable failover.

The following example shows failover with client-side load balancing disabled:

```
Properties prop = new Properties();
String url = "jdbc:oracle:thin:@ldap:ldap1.example.com:3500/
cn=salesdept,cn=OracleContext,dc=com/salesdb" +
"ldap:ldap2.example.com:3500/cn=salesdept,cn=OracleContext,dc=com/salesdb " +
"ldap:ldap3.example.com:3500/cn=salesdept,cn=OracleContext,dc=com/salesdb";

prop.put("oracle.net.ldap_loadbalance", "OFF" );
OracleDataSource ods = new OracleDataSource();
ods.setURL(url);
ods.setConnectionProperties(prop);
```

The JDBC Thin driver supports LDAP non-anonymous bind. A set of JNDI environment properties, which contains authentication information, can be specified for a data source. If an LDAP server is configured as not to allow anonymous bind, then you must provide the authentication information to connect to the LDAP server. The following example shows a simple clear-text password authentication:

```
String url = "jdbc:oracle:thin:@ldap:ldap.example.com:7777/
sales,cn=salesdept,cn=OracleContext,dc=com";

Properties prop = new Properties();
prop.put("oracle.net.ldap.security.authentication", "simple");
prop.put("oracle.net.ldap.security.principal","cn=salesdept,cn=OracleContext,dc=com");
prop.put("oracle.net.ldap.security.credentials", "mysecret");

OracleDataSource ods = new OracleDataSource();
ods.setURL(url);
ods.setConnectionProperties(prop);
```

In the preceding example, as JDBC passes down the three properties to JNDI, the authentication mechanism chosen by client is consistent with how these properties are interpreted by JNDI. For example, if the client specifies authentication information without explicitly specifying the `java.naming.security.authentication` property, then the default authentication mechanism is `simple`.

Starting from Oracle Database Release 21c, you can specify the location of the wallet that the driver uses for TLS negotiation with the LDAP server. You can add the user name and the password used for authentication to the wallet secret store.

> **See Also:**
>
> JDBC Java API Reference

8.2.9.1 Support for OpenLDAP

Starting from Oracle Database Release 21c, the JDBC Thin driver supports OpenLDAP.

> **See Also:**
>
> The OpenLDAP Main Page for more information about OpenLDAP