2 Getting Started

The following sections are included in this chapter:

2.1 Requirements for using UCP

This section describes the design-time and run-time requirements of UCP.

- JRE 8 or higher
- A JDBC driver or a connection factory class capable of returning a `java.sql.Connection` and `javax.sql.XAConnection` object

> **Note:**
>
> Oracle drivers from releases 11.2.0.4 or higher are supported. Advanced Oracle Database features, such as Oracle RAC and Fast Connection Failover, require the Oracle Notification Service library (`ons.jar`) that is included with the Oracle Client software.

- The `ucp.jar` library included in the classpath of the application
- The `ojdbc8.jar` library or the `ojdbc11.jar` library is included in the classpath of the application

> **Note:**
>
> Even if you use UCP with a third-party database and driver, you *must* use the Oracle `ojdbc8.jar` library or the `ojdbc11.jar` library because UCP has dependencies on this library.

- A database that supports SQL. Advanced features, such as Oracle RAC and Fast Connection Failover, require an Oracle Database.

2.2 Basic Connection Steps in UCP

UCP provides a pool-enabled data source that is used by applications to borrow connections from a UCP JDBC connection pool. A connection pool is not explicitly defined for the most basic use case. Instead, a default connection pool is implicitly created when the connection is borrowed.

The following steps describe how to get a connection from a UCP pool-enabled data source in order to access a database. The complete example is provided in Example 2-1:

1. Use the UCP data source factory (`oracle.ucp.jdbc.PoolDataSourceFactory`) to get an instance of a pool-enabled data source using the `getPoolDataSource` method. The data source instance must be of the type `PoolDataSource`. For example:

   ```
   PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();
   ```

2. Set the connection properties that are required to get a physical connection to a database. These properties are set on the data source instance and include: the URL, the user name, and password to connect to the database and the connection factory used to get the physical connection. These properties are specific to a JDBC driver and database. For example:

   ```
   pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
   pds.setURL("jdbc:oracle:thin:@//localhost:1521/orcl");
   pds.setUser("<user>");
   pds.setPassword("<password>");
   ```

3. Set any pool properties in order to override the connection pool's default behavior. the pool properties are set on the data source instance. For example:

   ```
   pds.setInitialPoolSize(5);
   ```

4. Get a connection using the data source instance. The returned connection is a logical handle to a physical connection in the data source's connection pool. For example:

   ```
   Connection conn = pds.getConnection();
   ```

5. Use the connection to perform some work on the database:

   ```
   Statement stmt = conn.createStatement ();
   stmt.execute("SELECT * FROM foo");
   ```

6. Close the connection and return it to the pool.

   ```
   conn.close();
   ```

2.2.1 Authentication in UCP

UCP provides transparent authentication, that is, the `PoolDataSource` behaves in the same way as the JDBC driver data source, while authenticating a connection.

UCP supports all the following authentication methods that the JDBC thin or the JDBC OCI driver suggests, and delegates any authentication action to the underlying driver:

- Authentication through passwords stored in Oracle Wallets
- Authentication using Kerberos
- Authentication through SSL certificates
- Authentication using Lightweight Directory Access Protocol (LDAP)

2.2.2 Authentication Using IAM Database Access Tokens in Oracle Cloud Infrastructure

In Oracle Database release 21.4 (21.4.0.0.1), the JDBC Thin drivers can access Oracle Autonomous Database on Shared Exadata Infrastructure, using a database access token generated by the Identity and Access Management (IAM) Cloud Service. UCP supports this authentication type using the `PoolDataSource.setTokenSupplier(Supplier)` method.

See Also:

Support for IAM Token-Based Authentication in Oracle Cloud Infrastructure

2.3 UCP API Overview

This section provides a quick overview of the most commonly used packages of the UCP API.

See Also:

Oracle Universal Connection Pool Java API Reference for complete details on the API.

oracle.ucp.jdbc

This package includes various interfaces and classes that are used by applications to work with JDBC connections and a connection pool. Among the interfaces found in this package, the `PoolDataSource` and `PoolXADataSource` data source interfaces are used by an application to get connections as well as get and set connection pool properties. Data source instances implementing these two interfaces automatically create a connection pool.

oracle.ucp.admin

This package includes interfaces for using a connection pool manager as well as MBeans that allow users to access connection pool and the connection pool manager operations and attributes using JMX operations. Among the interfaces, the `UniversalConnectionPoolManager` interface provides methods for creating and maintaining connection pool instances.

oracle.ucp

This package includes both required and optional callback interfaces that are used to implement connection pool features. For example, the `ConnectionAffinityCallback` interface is used to create a callback that enables or disables connection affinity and can also be used to customize connection affinity behavior. This package also contains statistics classes, UCP specific exception classes, and the logic to use the UCP directly, without using data sources.

2.4 UCP System Properties

For a detailed list of the UCP system properties, refer to the UCP Java API Reference.

See Also:

Oracle Universal Connection Pool Java API Reference

2.5 Basic Connection Example Using UCP

The following example is a program that connects to a database to do some work and then exits. The example is simple and in some cases not very practical; however, it does demonstrate the basic steps required to get a connection from a UCP pooled-enabled data source in order to access a database.

Example 2-1 Basic Connection Example

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import oracle.ucp.jdbc.PoolDataSourceFactory;
import oracle.ucp.jdbc.PoolDataSource;

public class BasicConnectionExample {
   public static void main(String args[]) throws SQLException {
      try
      {
         //Create pool-enabled data source instance.

         PoolDataSource  pds = PoolDataSourceFactory.getPoolDataSource();

         //set the connection properties on the data source.

         pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
         pds.setURL("jdbc:oracle:thin:@//localhost:1521/orcl");
         pds.setUser("<user>");
         pds.setPassword("<password>");

         //Override any pool properties.

         pds.setInitialPoolSize(5);

         //Get a database connection from the datasource.

         Connection conn = pds.getConnection();

         System.out.println("\nConnection obtained from " +
          "UniversalConnectionPool\n");

         //do some work with the connection.
         Statement stmt = conn.createStatement();
         stmt.execute("select * from foo");

         //Close the Connection.

         conn.close();
         conn=null;

         System.out.println("Connection returned to the " +
          "UniversalConnectionPool\n");

      }
      catch(SQLException e)
      {
         System.out.println("BasicConnectionExample - " +
          "main()-SQLException occurred : "
              + e.getMessage());
      }
```

```
    }
}
```

2.6 Minimal Pool configuration

You can configure a pool minimally and make use of the default values.

If you want to start a new pool data source, then instantiate it and set the following mandatory properties:

- The connection factory class, which is usually a JDBC driver data source, for example, `oracle.jdbc.pool.OracleDataSource`
- The peer database URL
- User name
- Password

All the other pool data source properties are optional. With this minimal configuration, the default pool size is as follows:

- Minimal pool size is 1
- Maximum pool size is `Integer.MAX_VALUE` (2147483647 by default)
- Initial pool size is 0

With the minimal configuration, the pool data source performs some minimal validation while borrowing the connections. The default value of the `setSecondsToTrustIdleConnection(int)` method is set to 120 seconds and all the timeouts are disabled.

See Also:

Overview of Validating Connections in UCP