11
UCP Asynchronous Extension

Starting from Oracle Database Release 23ai, UCP provides asynchronous extension. The asynchronous extension is a set of methods that extend the UCP standard to offer asynchronous database access.

The asynchronous extension is integrated into UCP in such a way that you can use all the features of UCP with this extension, with some changes in your source code. This chapter describes the changes that you must make to your code for using UCP in an asynchronous way.

> **Note:**
>
> This feature is supported starting with JDK 11.

This chapter covers the following topics:

11.1 Overview of UCP Asynchronous Extension

The UCP asynchronous extension uses non-blocking mechanisms for creating connection objects, so your application immediately receives either a `CompletableFuture` or a `Publisher` of a connection to be borrowed.

You must perform the following to achieve this:

1. Instantiate a Connection Builder.
2. Create a connection asynchronously with `UCPConnectionBuilder` using either a `CompletableFuture<Connection>` or a `Publisher<Connection>`.

The asynchronous extension also lets you borrow XA connections with a `UCPXAConnectionBuilder` using either a `CompletableFuture<XAConnection>` or a `Publisher<XAConnection>`.

When an asynchronous method is called, it performs as much work as possible on the calling thread, without blocking on a network read or write. An asynchronous method call returns immediately after a request is about to be written to the network, without waiting for a response. When I/O readiness is detected for a network channel, the polling thread arranges for a worker thread to handle the event. The worker thread reads from the network and then notifies a `CompletableFuture` or a `Publisher` that an operation is complete. Upon notification, the `CompletableFuture` or `Publisher` arranges worker threads that emit a signal to each of its `Subscribers`.

The `java.util.concurrent.Executor` interface manages the worker threads, while the default `Executor` is the `java.util.concurrent.ForkJoinPool.commonPool` method. If you do not implement the `executor()` code in your application source code, then the asynchronous borrow operation runs with the default `ForkJoinPool` executor. For setting an arbitrary executor to serve an asynchronous borrow, you can use the `executor(executor)` call.

> **See Also:**
>
> Overview of JDBC Reactive extensions

11.2 Example: UCP Asynchronous Extension

This section lists a few examples that demonstrate how to use UCP asynchronous extension.

Example 11-1 Creating a Connection Asynchronously with CompletableFuture<Connection>

```
...
final PoolDataSource pds = new PoolDataSourceImpl();
[//Initialize PoolDataSource object in the standard way]

final CompletionStage<Connection> connectionStage =
  pds.createConnectionBuilder()
    .user(<user name>)
    .password(<password>)
    .executor(executor)
    .buildAsyncOracle();

final CompletionStage<String> queryStage =
  connectionStage.thenApply(connection -> {
     [//Perform operations on the connection]
  });
...
```

Example 11-2 Creating a Connection Asynchronously with Publisher<Connection>

```
...
final PoolDataSource pds = new PoolDataSourceImpl();
[//Initialize PoolDataSource object in the standard way]

final Publisher<Connection> connectionPublisher =
  pds.createConnectionBuilder()
    .user(<user name>)
    .password(<password>)
    .executor(executor)
    .buildConnectionPublisherOracle();

[//Perform standard activities on the Publisher]
...
```

Example 11-3 Creating an XA Connection Asynchronously with UCPXAConnectionBuilder

```
...
final PoolXADataSource pds = new PoolXADataSourceImpl();
[//Initialize PoolXADataSource object in the standard way]

final CompletionStage<XAConnection> connectionStage =
```

```
  pds.createXAConnectionBuilder()
    .user(<user name>)
    .password(<password>)
    .executor(executor)
    .buildAsyncOracle();

final CompletionStage<String> queryStage =
  connectionStage.thenApply(xaConnection -> {
     [//Perform operations on the XAConnection]
  });
...
```

Example 11-4 Creating an XA Connection Asynchronously with Publisher<XAConnection>

```
...
final PoolXADataSource pds = new PoolXADataSourceImpl();
[//Initialize PoolXADataSource object in the standard way]

final Publisher<XAConnection> xaConnectionPublisher =
  pds.createXAConnectionBuilder()
    .user(<user name>)
    .password(<password>)
    .executor(executor)
    .buildConnectionPublisherOracle();

[//Perform standard activities on the Publisher]
...
```

11.3 Asynchronous Connection Labeling

You can take advantage of the UCP connection labeling feature even in the asynchronous mode. For achieving this, you must override the default version of the new `oracle.ucp.ConnectionLabelingCallback.configureAsync()` method.

configureAsync: Asynchronous Version of the Configure Method

For standard connection labeling, you use the `configure` method in your application. For using connection labeling in the asynchronous mode, you must use the `configureAsync` method. The definition of the `configureAsync` method is as follows:

```
default CompletionStage<Boolean> configureAsync(Properties requestedLabels,
Object connection) {
  throw new NoSuchMethodError();
```

> **See Also:**
>
> - Labeling Connections in UCP
> - *Oracle Universal Connection Pool Java API Reference*

11.4 Example: Asynchronous Connection Labeling

This section contains an examples that demonstrates how to use UCP asynchronous extension with connection labeling.

Example 11-5 Creating a Connection Asynchronously with Connection Labeling

```
package tests.ucp.async.labeling;

import oracle.ucp.ConnectionLabelingCallback;
import oracle.ucp.jdbc.PoolDataSource;
import oracle.ucp.jdbc.PoolDataSourceImpl;

import java.sql.Connection;
import java.util.Properties;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.CompletionStage;

public class AsyncLabelingExample {
  public static void main(String ... args) throws Exception {
    final PoolDataSource pds = new PoolDataSourceImpl();

    // Set the pool data source properties

    final ConnectionLabelingCallback labelingCallback = new
ConnectionLabelingCallback() {
      @Override
      public int cost(Properties requestedLabels, Properties currentLabels) {
        // some cost manipulations, same as in synchronous case
        return 0; // or some other integer, depending on the cost computation
logic
      }

      @Override
      public boolean configure(Properties requestedLabels, Object connection)
{
        // some connection configuration manipulations for synchronous case,
        // not used in asynchronous case, so it can be skipped.
        return true;
      }

      @Override
      public CompletionStage<Boolean> configureAsync(Properties
requestedLabels, Object connection) {
        final var cf = new CompletableFuture<Boolean>();

        // Perform some asynchronous connection configuration
        doSomeConfigAction((Connection)connection).whenComplete((p, e) -> {
          if (null == e) {
            cf.complete(p);
          } else {
            cf.completeExceptionally(e);
          }
        });
```

```
        return cf;
      }

      private CompletableFuture<Boolean> doSomeConfigAction(Connection conn) {
        final var cf = new CompletableFuture<Boolean>();

        // ...
        // configure the connection asynchronously
        // ... complete CompletableFuture with result or exception ...

        return cf;
      }
    };

    pds.registerConnectionLabelingCallback(labelingCallback);

    // some labeling code

  }
}
```