14 Diagnosing a Connection Pool

The following parameters are used for diagnosing Universal Connection Pool (UCP):

- Pool Statistics
- Dynamic Monitoring Service Metrics
- Overview of Logging and Tracing in UCP
- About Viewing Oracle RAC Statistics
- Exceptions and Error Codes

14.1 Pool Statistics

Universal Connection Pool (UCP) provides a set of run-time statistics for the connection pool. These statistics can be divided into the following two categories:

- Noncumulative

  These statistics apply only to the current running connection pool instance.
- Cumulative

  These statistics are collected across multiple pool start/stop cycles.

The `oracle.ucp.UniversalConnectionPoolStatistics` interface provides methods that are used to query the connection pool statistics. The methods of this interface can be called from a pool-enabled data source and pool-enabled XA data source, using the `oracle.ucp.jdbc.PoolDataSource.getStatistics` method. For example:

```
PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
...
...
int totalConnsCount = pds.getStatistics().getTotalConnectionsCount();
System.out.println("The total connetion count in the pool is "+ totalConnsCount +".");
```

The `oracle.ucp.jdbc.PoolDataSource.getStatistics` method can also be called by itself to return all connection pool statistics as a `String`.

14.2 Dynamic Monitoring Service Metrics

UCP supports all the pool statistics to be in the form of Dynamic Monitoring Service (DMS) metrics. You must include the `dms.jar` file in the class path of the application to collect and utilize these DMS metrics.

UCP supports DMS metrics collection in both the pool manager interface and the pool manager MBean. You can use the `UnversalConnectionPoolManager.startMetricsCollection` method to start collecting DMS metrics for the specified connection pool instance, and use the `UnversalConnectionPoolManager.stopMetricsCollection` method to stop DMS metrics collection. The metrics update interval can be specified using the

`UnversalConnectionPoolManager.setMetricUpdateInterval` method. The pool manager MBean exports similar operations.

14.3 Overview of Logging and Tracing in UCP

Two major aspects of UCP diagnosability are logging and tracing.

Logging is writing log records into a console, a file, or any other log handler that is defined by standard logging properties. The name of a logger is the same as a pool data source name. A pool data source name is set with an appropriate property. You can leave it unnamed as well, in which case, all diagnostics goes in the common logger. The name of the common logger is `oracle.ucp`. Logging implements the Java Logging API, `java.util.logging`.

Tracing is the special logging use case, where a log record is written into an in-memory ring buffer until an event triggers UCP Diagnosability to dump that buffer into a corresponding logger. UCP has the following two categories of trace buffers:

> **Note:**
>
> A pool data source and a UCP have a one-to-one mapping.

- A category that consists of one common buffer, which is used for tracing logs from the static methods that are not directly related to a pool with a designated name. The common buffer is permanent for the complete life span of the associated UCP.
- A category that contains as many buffers as the number of named Pool Data Source (PDS) objects, where every buffer is mapped to a single named PDS. Every trace from that PDS goes into the corresponding buffer. Its life starts from a pool start up and ends with a pool destruction.

14.3.1 Logging and Tracing Settings

When you enable logging, you should not enable tracing because the log records get immediately dumped into a logger in such a case, and there is no need to duplicate log records into a trace buffer. So, if logging is on, regardless of the tracing setting, tracing gets disabled automatically.

> **Note:**
>
> By default, tracing is on and logging is off.

The following table summarizes this functionality:

Logging Setting	Tracing Setting	Logging Functionality	Tracing Functionality
`off`	`off`	Disabled	Disabled
`on`	`off`	Enabled	Disabled
`off`	`on`	Disabled	Enabled
`on`	`on`	Enabled	Disabled

14.3.2 Diagnosability System Properties and Command Line

You can set the initial diagnosability properties as JVM system properties.
You can achieve this in the following ways:

- Using the Oracle JVM command line
- In your application source code
- Using Java Management Extensions (JMX)

Property Name	Description
`oracle.ucp.diagnostic.enableTrace`	Is the flag to enable tracing. The default value of this property is `true`.
`oracle.ucp.diagnostic.enableLogging`	Is the flag to enable debug logging. The default value of this property is `false`.
`oracle.ucp.diagnostic.bufferSize`	Specifies the in-memory trace buffer size. The default value is 1024.
`oracle.ucp.diagnostic.loggingLevel`	Specifies the default logging level that is used if no other value is specified for a logger in a logging configuration file. The default value of this property is `INFO`.
`java.util.logging.config.file`	Specifies the logging property file name to load. The default value is `ConsoleHandler` for all loggers.
`oracle.ucp.diagnostic.errorCodesToWatchList`	Provides the list of exceptions and error codes that triggers dumping of all traces into their loggers.

14.3.3 Logging Configuration File

The logging configuration file lets you configure your logging settings as described in the `java.util.logging` package. If you want to set different levels for different loggers, then you must set those in the logging configuration file.
Perform the following for configuring the common logger:

- Specify the logger as `oracle.ucp`.
- Set an appropriate logging level.
- Specify where and how to write the logs. For example, whether the logs should be written to the console or to a file. In case of a file, specify the file name, formatter, and so on.

Perform the following for configuring a logger that is specific to a pool data source object:

- Set a level for the logger with the name of that pool data source.
- Set an appropriate logging level.

> **Note:**
>
> If you do not specify the level for a logger, then a default level, which is set in the `ucp.diagnostic.loggingLevel`, is used.

Example 14-1 Logging Configuration File

The following is an example of the UCP diagnosability logging configuration file:

```
oracle.ucp.level=FINE
handlers = java.util.logging.FileHandler
java.util.logging.FileHandler.pattern = ./test.log
java.util.logging.FileHandler.limit = 0
java.util.logging.FileHandler.count = 1
java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
pool-name1=FINEST
pool-name2=SEVERE
```

Logging Level

Logging levels in the UCP Diagnosability framework are the same as the ones available in the `java.util.logging` package, but UCP also supports a numeric logging level in the range from `Integer.MIN_VALUE` to `Integer.MAX_VALUE`. The following table lists all the valid logging levels:

Logging Level	Value
`OFF`	`Integer.MAX_VALUE`
`SEVERE`	`1000`
`WARNING`	`900`
`INFO`	`800`
`CONFIG`	`700`
`FINE`	`500`
`FINER`	`400`
`FINEST`	`300`
`ALL`	`Integer.MIN_VALUE`

14.3.4 Tracing the Error Codes to Watch

The error codes to watch are applicable only for tracing. If tracing is disabled, then the error codes to watch setting is ignored.
There are two event types that trigger the trace buffers to be dumped into an appropriate logger:

- Any log message with a `SEVERE` log level.
- A few subset of exceptions or errors, for example, the subclasses of the `java.lang.Throwable` class, which are thrown by the JDBC driver or UCP. These errors or exceptions are then caught internally and logged with the `WARNING` level. The subset of these exceptions are defined with the `ucp.diagnostic.errorCodesToWatchList` property.

A `WARNING` log message, which contains a subclass of `java.lang.Throwable`, can cause dumping of a trace buffer into a logger, if the exception is found in a list of error codes. If an exception contains a stack of errors or exceptions, then UCP traverses through that stack with an attempt to find a matching exception and the corresponding error code, if applicable. The format of the list of error code is as follows:

```
["<subclass1>.<Exception1>:111,222,333",
"<subclass2>.<Exception2>:444,555,666", "<subclass3>.<Exception3>"]
```

where, `subclass1.Exception1`, `subclass2.Exception2`, and `subclass3.Exception3` are the `java.lang.Throwable` subclasses.

For example, if you see an error code like the following:

```
[ "oracle.ucp.UniversalConnectionPoolException:45054,45065,45067",
"java.sql.SQLException:12521,12514,12757,12523",
"java.lang.IllegalStateException", "java.lang.NullPointerException" ]
```

It means that the tracing is triggered in the following sequence:

1. `oracle.ucp.UniversalConnectionPoolException` with the vendor error codes 45054, 45065, and 45067
2. `java.sql.SQLException` with the vendor error codes 12521, 12514, 12757, and 12523
3. `IllegalStateException`
4. `NullPointerException`

Only fully-qualified `java.lang.Throwable` subclass names can cause dumping of a trace buffer into a logger. For example, `java.sql.NullPointerException` is a valid name, while `NullP`, `NullPointerException`, `["12154"]`, `["ORA-"]`, or `["ORA-12154"]` is not. If specified wrongly, then the subclass name causes parsing and/or class resolution errors logged, along with a `WARNING`. Any number conversion error too can cause a parsing error logged, along with a `WARNING`.

In case of parse and class resolution failures, every `WARNING` log record, with any exception, causes dumping a trace buffer. This is regardless of exception types and designated error codes. Once error codes are ignored in logging mode, they are not parsed anymore.

Comma-separated lists of error codes are used only in case of the following two classes and their subclasses:

- `java.sql.SQLException` and its subclasses like `java.sql.SQLRecoverableException` or `java.sql.SQLSyntaxError`
- `oracle.ucp.UniversalConnectionPoolException` and its subclasses like `NoAvailableConnectionsException`

If you specify comma-separated lists of error codes with other exception types, then those error codes are ignored. With an empty list of error codes for the two exceptions and their subclasses mentioned earlier, all error codes are applicable. The UCP Diagnosability has a default error codes list. Refer to "UCP Exception Error Codes" for the list of `oracle.ucp.UniversalConnectionPoolException` error codes.

If you want the trace buffer to be dumped by any `WARNING` message with an exception, then you can perform it in the following way:

Example 14-2   To dump traces by any exception

```
"["java.lang.Throwable"]"
```

14.3.5 MBeans for UCP Diagnosability

UCP provides two MBeans for UCP diagnosability.

- `ucp.admin`: It contains attributes amd operations for the whole JVM. It manages the buffer associated with all the pool data sources, and also the common in-memory buffer. You can change the initial values of the following MBean attributes to set it:
  - `enableTrace`
  - `enableLogging`
  - `loggingLevel`
  - `bufferSize`
  - `errorCodesToWatchList`
  - `loggingConfigFileName`

  Also, there is an operation named `dumpInMemoryTrace`. Launching this operation, you can dump contents of all in-memory buffers into their appropriate loggers.
- `ucp.admin.UniversalConnectionPoolMBean`: It contains a tree of existing pool insntances, where every instance has its own attribute to modify an appropriate property of that specific pool. The attributes are the following:
  - `enableTrace`
  - `enableLogging`
  - `loggingLevel`
  - `inMemoryTraceSize`

14.4 About Viewing Oracle RAC Statistics

UCP provides a set of Oracle RAC run-time statistics that are used to determine how well a connection pool is utilizing Oracle RAC features and are also used to help determine whether the connection pool has been configured properly to use the Oracle RAC features. The statistics report FCF processing information, run-time connection load balance success/failure rate, and affinity context success/failure rate.

The `OracleJDBCConnectionPoolStatistics` interface that is located in the `oracle.ucp.jdbc.oracle` package provides methods that are used to query the connection pool for Oracle RAC statistics. The methods of this interface can be called from a pool-enabled and pool-enabled XA data source using the data source's `getStatistics` method. For example:

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

Long rclbS = ((OracleJDBCConnectionPoolStatistics)pds.getStatistics()).
   getSuccessfulRCLBBasedBorrowCount();
System.out.println("The RCLB success rate is "+rclbS+".");
```

The data source's `getStatistics` method can also be called by itself and returns all connection pool statistics as a `String` and includes the Oracle RAC statistics.

14.4.1 Fast Connection Failover Statistics

The `getFCFProcessingInfo` method provides information on recent Fast Connection Failover (FCF) attempts in the form of a `String`. The FCF information is typically used to help diagnose FCF problems. The information includes the outcome of each FCF attempt (successful or failed), the relevant Oracle RAC instances, the number of connections that were cleaned up, the exception that triggered the FCF attempt failure, and more. The following example demonstrates using the `getFCFProcessingInfo` method:

```
Sting fcfInfo = ((OracleJDBCConnectionPoolStatistics)pds.getStatistics()).
   getFCFProcessingInfo();
System.out.println("The FCF information: "+fcfInfo+".");
```

Following is a sample output string from the `getFCFProcessingInfo()` method:

```
    Oct 28, 2008 12:34:02 SUCCESS <Reason:planned> <Type:SERVICE_UP> \
      <Service:"svvc1"> <Instance:"inst1"> <Db:"db1"> \
      Connections:(Available=6 Affected=2 FailedToProcess=0 MarkedDown=2 Closed=2) \
      (Borrowed=6 Affected=2 FailedToProcess=0 MarkedDown=2 MarkedDeferredClose=0 
Closed=2) \
      TornDown=2 MarkedToClose=2 Cardinality=2
    ...
    Oct 28, 2008 12:09:52 SUCCESS <Reason:unplanned> <Type:SERVICE_DOWN> \
      <Service:"svc1"> <Instance:"inst1"> <Db:"db1"> \
      Connections:(Available=6 Affected=2 FailedToProcess=0 MarkedDown=2 Closed=2) \
      (Borrowed=6 Affected=2 FailedToProcess=0 MarkedDown=2 MarkedDeferredClose=0 
Closed=2)
    ...
    Oct 28, 2008 11:14:53 FAILURE <Type:HOST_DOWN> <Host:"host1"> \
      Connections:(Available=6 Affected=4 FailedToProcess=0 MarkedDown=4 Closed=4) \
      (Borrowed=6 Affected=4 FailedToProcess=0 MarkedDown=4 MarkedDeferredClose=0 
Closed=4)
```

If you enable logging, then the preceding information will also be available in the UCP logs and you will be able to verify the FCF outcome.

14.4.2 Run-Time Connection Load Balance Statistics

The run-time connection load balance statistics are used to determine if a connection pool is effectively utilizing the run-time connection load balancing feature of Oracle RAC. The statistics report how many requests successfully used the run-time connection load balancing algorithms and how many requests failed to use the algorithms. The `getSuccessfulRCLBBasedBorrowCount` method and the `getFailedRCLBBasedBorrowCount` method, respectively, are used to get the statistics. The following example demonstrates using the `getFailedRCLBBasedBorrowCount` method:

```
Long rclbF = ((OracleJDBCConnectionPoolStatistics)pds.getStatistics()).
   getFailedRCLBBasedBorrowCount();
System.out.println("The RCLB failure rate is: "+rclbF+".");
```

A high failure rate may indicate that the Oracle RAC Load Balancing Advisory or connection pool is not configured properly.

14.4.3 Connection Affinity Statistics

The connection affinity statistics are used to determine if a connection pools is effectively utilizing connection affinity. The statistics report the number of borrow requests that succeeded in matching the affinity context and how many requests failed to match the affinity context. The `getSuccessfulAffinityBasedBorrowCount` method and the `getFailedAffinityBasedBorrowCount` method, respectively, are used to get the statistics. The following example demonstrates using the `getFailedAffinityBasedBorrowCount` method:

```
Long affF = ((OracleJDBCConnectionPoolStatistics)pds.getStatistics()).
   getFailedAffinityBasedBorrowCount();
System.out.println("The connection affinity failure rate is: "+affF+".");
```

14.5 Exceptions and Error Codes

Many UCP methods throw the `UniversalConnectionPoolException`, with exception chaining supported. You can call the `printStackTrace` method on the thrown exception, to identify the root cause of the exception. The `UniversalConnectionPoolException` includes standard Oracle error codes that are in the range of 45000 and 45499. The `getErrorCode` method can be used to retrieve the error code for an exception.